Bayesian Hyperparameter Optimization

Relational and Scalable Surrogate Models for Hyperparameter Optimization Across Problem Instances

By

NICOLAS SCHILLING

Information Systems and Machine Learning Lab
UNIVERSITY OF HILDESHEIM

A dissertation submitted to the University of Hildesheim for the degree of *Doctor of Natural Sciences* (Dr. rer. nat.) in the subject of computer science

HILDESHEIM, JUNE 2018

Abstract

Machine learning is often confronted with the problem of learning prediction models on a set of observed data points. Given an expressive data set of the problem to solve, using powerful models and learning algorithms is only hindered by setting the right configurations for both. Unfortunately, the magnitude of the performance difference is large, which makes choosing right configurations an additional problem that is only solved by experienced practitioners.

In this thesis, we will address the problem of hyperparameter optimization for machine learning and present ways to solve it. We firstly introduce the problem of supervised machine learning. We then discuss many examples of hyperparameter configurations that can be considered prior to learning the model. Afterwards, we introduce methods on finding the right configurations, especially those methods that work in the scheme of Bayesian optimization, which is a framework for optimizing black-box functions. Black-boxes are functions where for a given input one can only observe an output after running a costly procedure. Usually, in black-box optimization so-called surrogate models are learned to reconstruct the observations to then offer a prediction for unobserved configurations. Fortunately, recent outcomes show that transfering the knowledge across problems, for example by learning surrogates across different data sets being solved by the same model class, shows promising results.

We tackle the problem of hyperparameter optimization in mainly two different ways. At first, we consider the problem of hyperparameter optimization as a recommendation problem, where we want to learn data set features as well as their interaction with the hyperparameter configurations as latent features in a factorization based approach. We build a surrogate model that is inspired by the complexity of neural networks as well as the ability to learn latent embeddings as in factorization machines. Secondly, as the amount of meta knowledge increases every day, surrogate models need to be scalable. We consider Gaussian processes, as they themselves are hyperparameter free and work very well in most hyperparameter optimization cases. Unfortunately, they are not scalable, as a matrix in the size of the number of data points has to be inverted for inference. We show various methods of simplifying a Gaussian process by using an ensemble of Gaussian process experts, which is much faster to learn due to its paralellization properties while still showing very competitive performance.

We conclude the thesis by discussing the aspect of learning across problems in more detail than simply learning across different data sets. By learning hyperparameter performance across different models, we show that also model choice can be handled by the proposed algorithms. Additionally, we show that hyperparameter performance can even be transfered across different problem tasks, for example from classification to regression.

Zusammenfassung

Maschinelles Lernen beschäftigt sich hauptsächlich damit, Modelle auf bereits observierten Daten zu lernen. Das einzige Hindernis - sofern aussagekräftige Daten gegeben sind - ist dabei ist üblicherweise die richtige Konfiguration des Modells und des Lernalgorithmusses zu finden. Leider hängt die Güte des gelernten Modells sehr von der gewählten Konfiguration ab, sodass diese Aufgabe üblicherweise von erfahrenen Anwendern bewältigt wird.

In dieser Dissertation adressieren wir das Problem der Hyperparameteroptimierung im Maschinellen Lernen. Als Erstes geben wir eine Einführung in überwachtes Maschinelles Lernen und erörtern dabei Hyperparameter welche vor dem Lernprozess gewählt werden müssen. Danach beschreiben wir Methoden der Hyperparameteroptimierung, insbesondere solche die auf die Bayessche Optimierung beruhen, welches oft bei der Optimierung von Black-Box Funktionen angewandt wird. Eine Black-Box ist eine Funktion, die zu einem gegebenen Eingabe- einen festen Ausgabewert liefert, dessen Evaluation aber teuer ist. Um eine Black-Box zu optimieren werden üblicherweise sogenannte Surrogatmodelle auf den bisherigen Observationen gelernt, um dann für neue Konfigurationen eine Vorhersage zu tätigen. Glücklicherweise zeigen Arbeiten der jüngeren Vergangenheit, dass Wissen über die Performanz von einzelnen Hyperparameterkonfigurationen über Datensätze hinaus transferiert werden kann.

Wir lösen das Problem der Hyperparameteroptimierung auf zwei Weisen. Zunächst verstehen wir es als ein Empfehlungssystem, wo wir für jede kategorische Variable, wie z.B. dem Datensatz, latente Charakteristiken von Datensätzen und Hyperparametern in einem Faktorisierungsmodell lernen. Dazu schlagen wir ein Surrogatmodell vor, welches die Komplexität von neuronalen Netzen besitzt, aber dennoch latente Charakteristiken lernen kann wie in einer Faktorisierungsmaschine. Weiterhin befassen wir uns mit dem Problem der Skalierbarkeit von Surrogatmodellen, da prinzipiell die Menge an Metawissen jeden Tag beständig wächst. Wir nutzen Gauss-Prozesse, da diese selbst keine Hyperparameter besitzen, bzw. diese gelernt werden können. Leider sind Gauss-Prozesse nicht skalierbar, da bei der Inferenz eine Matrix in der Größe des Datensatzes invertiert werden muss. Wir schlagen daher vor, den Gauss-Prozess als Produkt von einzelnen Gauss-Prozessen, sogenannten Experten, zu lernen, was aufgrund der Parallelisierbarkeit viel schneller möglich ist und besser skaliert.

Abschliessend betrachten wir das Problem des Lernens über Problemaspekte genauer, indem wir beispielsweise zeigen, dass unsere Surrogatmodelle auch dafür genutzt werden können um automatische Modellwahl durchzuführen. Darüber hinaus zeigen wir, dass Hyperparameterperformanz auch über verschiedene Aufgabenstellungen gelernt werden kann, beispielsweise von der Klassifikation zur Regression.

Acknowledgements

I believe that there is honestly not enough space for me to write about all the people and all their actions that helped me so much before and during my PhD studies. I will, however, try to give a big thanks to all that supported me throughout my time in Hildesheim, but also before.

First of all, I want to thank my supervisor, Lars Schmidt-Thieme, who offered me the opportunity to work and study Machine Learning in his group, the Information Systems and Machine Learning Lab at the University of Hildesheim. Lars, I will never forget all the reading groups where you explained something at the board and suddenly all clicked into place. Thank you so much for all your support and meaningful insight into all the challenging problems.

Secondly, I want to give thanks to my girlfriend Christin Küpper, whose support has proven to be indispensable for writing this thesis. Whenever I was afraid of gaining good paper results out of a new idea or not, I could talk to you, you helped me to always stay positive. Thank you so much for being in my life.

Next, I want to thank my parents - Renate and Ludger Schilling - for their exhaustlessness in helping and supporting me throughout my life and scientific career. You spent so much money and effort for my mathematics studies in Bremen and I am deeply thankful for that.

I also want to thank my brother Lennart and best friend Carsten Niermann for all your support throughout my life.

A very big thank you goes to Martin Wistuba whom I first met at Gästehaus Klocke in Hildesheim, prior to the Phd workshop which we both attended. When you saw me on the hallway your only question was: “Phd workshop?”. A question that I replied immediately with a “yes”, which was the start of both our scientific careers as well as a great friendship. Martin, working with you on our DFG project HyLAP has been an amazing time, going to ECML together two years in a row has made these conferences so enjoyable, and you have always been a colleague to discuss work-related issues as well as non work-related issues. It was only because of the discussions with you that I was able to do the research that I carried out in the end. Thank you, Martin!

Two other persons for whom I have to express my deepest thankfulness are Lucas Drumond and André Busche, who are both former colleagues at ISMLL. When I started, I had so many questions regarding everything, learning algorithms, factorization models, implementation details (yes I am talking about pointers) in Java, linux console and bash scripts that I needed to run to process the experiment results on the cluster, all of these

questions have been answered by both of you. You have always helped me so much in every aspect of my PhD, and I deeply thank you for that.

I want to also thank all other persons who I have met at ISMLL and who have helped me such as Kerstin, Jörg, Josif, Umer, Rasoul, Carlotta, Mohsan, Lydia, Nghia, Hanh and the more recent colleagues Rafael, Mofassir, Hadi, Ahmed, Randolf, Jonas, Eya and Shayan. Thank you all!

Finally, I want to thank Theodore Alexandrov, whom I first met as lecturer in a seminar called “Clustering and Classification” at the university of Bremen. Theodore has supervised my Diploma thesis on “Semi-Supervised Kernel-Based Learning” in Bremen and has motivated me to apply for PhD positions in the area of Machine Learning after I completed my thesis. It is indispensable to say that you have been the driving force in my decision to pursue a PhD and to be honest, it was simply a coincidence that positions were opened in Hildesheim just when I finished my diploma in mathematics.

Thank you all so much!

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

Table of Contents

Page

List of Tables **xiii**

List of Figures **xv**

1 Introduction **1**
- 1.1 Overview . . . 2
- 1.2 Main Contributions . . . 3
- 1.3 Published Work . . . 4

2 Hyperparameter Optimization in Machine Learning **9**
- 2.1 Supervised Machine Learning . . . 9
- 2.2 Occurences of Hyperparameters in Machine Learning . . . 11
  - 2.2.1 Objective Function . . . 12
  - 2.2.2 Learning Algorithm . . . 12
  - 2.2.3 Model Choice . . . 13
  - 2.2.4 Data Processing . . . 14
- 2.3 The Hyperparameter Optimization Problem . . . 14
  - 2.3.1 Hyperparameter Optimization without Meta Knowledge . . . 14
  - 2.3.2 Hyperparameter Optimization Across Data Sets . . . 16
- 2.4 A Word on Notation . . . 17

3 Bayesian Hyperparameter Optimization **19**
- 3.1 Bayesian Optimization . . . 19
- 3.2 Sequential Model-Based Optimization . . . 22
  - 3.2.1 Acquisition Functions . . . 25
- 3.3 Related Work . . . 30
  - 3.3.1 Other Hyperparameter Optimization Methods . . . 30

3.3.2 Hyperparameter Optimization based on SMBO 33

4 Experimental Setup **37**
4.1 Meta Data Creation 37
4.1.1 AdaBoost Meta Data 38
4.1.2 SVM Meta Data 39
4.1.3 Weka Meta Data 42
4.1.4 Multilayer Perceptron Meta Data 45
4.2 Meta Features 48
4.2.1 Task-dependent meta features 49
4.2.2 Data-dependent meta features 50
4.3 Evaluation Measures 51
4.3.1 Average Hyperparameter Rank 52
4.3.2 Average Rank among Competitors 53
4.3.3 Average Distance to the Minimum 54
4.3.4 Fraction of Unsolved Data Sets 54

5 Relational Hyperparameter Optimization with Factorized Multilayer Perceptrons **55**
5.1 Relational Models in Recommender Systems 56
5.1.1 Single Relational Matrix Factorization 57
5.1.2 Factorization Machines 58
5.2 Relational Models for Hyperparameter Optimization 59
5.2.1 Requirements for Cross-Data Surrogates 59
5.2.2 Factorization Machines as Surrogates 60
5.2.3 Multilayer Perceptrons as Surrogates 61
5.2.4 Factorized Multilayer Perceptrons 62
5.2.5 Estimating Predictive Posteriors for MLP and FMLP 64
5.3 Empirical Evaluation 71
5.3.1 Competing Surrogate and Regression Models 71
5.3.2 Reconstruction of Response Surfaces 72
5.3.3 Uncertainty Estimation 73
5.3.4 SMBO-based Hyperparameter Optimization 75

6 Scalable Hyperparameter Optimization with Gaussian Process Ensembles **79**

6.1 Gaussian Process Regression for Noisy Data 81
6.1.1 Gaussian Process Priors 81
6.1.2 Kernel Functions 83
6.1.3 Inference in Gaussian Processes 86
6.1.4 Efficiently Updating the Kernel Inverse Sequentially 88
6.1.5 Estimating Kernel Hyperparameters 89
6.2 Product of Experts Models for Hyperparameter Optimization 90
6.2.1 (Generalized) Product of Experts 91
6.2.2 (Robust) Bayesian Committee Machines 92
6.2.3 POE for Hyperparameter Optimization 94
6.2.4 Empirical Evaluation of POE-based Hyperparameter Optimization 95
6.3 Transfer Surrogate and Acquisition Framework 99
6.3.1 Transfer Surrogate Models 99
6.3.2 Transfer Acquisition Functions 102
6.4 Empirical Evaluation 104
6.4.1 Performance in Sequential Model-Based Optimization 104
6.4.2 Significance Analysis 111
6.4.3 Runtime 115

7 Hyperparameter Optimization Across Problem Instances 119
7.1 Combined Hyperparameter Optimization and Model Choice 120
7.1.1 Experimental Results 121
7.1.2 Model Choice 123
7.2 Hyperparameter Optimization Across Problem Tasks 124
7.2.1 Hyperparameter Optimization on All Tasks 125
7.2.2 Transfering Hyperparameter Performance Across Tasks 127

8 Conclusion 131

A Appendix 135
A.1 Multivariate Gaussian Distributions 135
A.1.1 Marginal of a Gaussian Distribution 137
A.1.2 Conditional of a Gaussian Distribution 138
A.1.3 Product of Gaussian Distributions 142

Bibliography 145

List of Tables

Table		Page
4.1	The grid used to create the Weka meta-data set (Part 1).	43
4.2	The grid used to create the Weka meta-data set (Part 2).	44
4.3	The grid used to create the Weka meta-data set (Part 3).	45
4.4	Overview of the hyperparameter grid used to create the MLP meta data set	48
4.5	The list of all meta-features used in our classification tasks.	49
4.6	The list of all meta-features used for the MLP meta data	51
5.1	Confidence intervals of the resulting RMSE for all models when reconstructing the response surface of SVM and AdaBoost	72

List of Figures

Figure **Page**

2.1 Response surface of a support vector machine on the Iris data set. 15

3.1 Overview of Bayesian optimization on a one-dimensional function 21
3.2 Expected Improvement is shown for both the uncertainty and the improvement 29

4.1 AdaBoost response surfaces for automobile (left) and pendigits (right). 39
4.2 AdaBoost response surfaces for bupa (left) and svmguide1 (right). 40
4.3 SVM response surfaces for the polynomial (left) kernel and the Gaussian (right) kernel for the ecoli data set. 41

5.1 Example of a neural network structure with three hidden layers. 62
5.2 Predictive Posterior of a multilayer perceptron. 69
5.3 Development of the average hyperparameter rank with increasing numbers of trials. 74
5.4 Development of the average rank for AdaBoost (left) and the SVM (right) meta data set over an increasing numbers of trials. 75
5.5 Development of the average hyperparameter rank for AdaBoost (left) and the SVM (right) meta data set over an increasing numbers of trials. 77

6.1 Comparison of candidate functions drawn from a Gaussian process prior using the SE and the Dirichlet kernel. 84
6.2 Comparison of candidate functions drawn from a Gaussian process prior using the linear and the polynomial kernel. 86
6.3 A GP posterior prediction. 87
6.4 Average Rank for AdaBoost and SVM. 96
6.5 Average Distante to Minimum for AdaBoost and SVM. 97
6.6 Runtime comparison among the most competitive surrogate models. 98

6.7 SGPT-R is outperforming the other two approaches due to the decaying influence of the meta-data for the task of hyperparameter optimization. 105
6.8 The adaptive weights also allows SGPT-R to outperform its variants for the task of combined algorithm selection and hyperparameter optimization. . . . 105
6.9 Performance evaluation of SGPT-R on SVM. 106
6.10 Performance evaluation of SGPT-R on Weka. 106
6.11 SGPT and TAF comparison on SVM 108
6.12 SGPT and TAF comparison on Weka 108
6.13 Performance of TAF on SVM 109
6.14 Performance of TAF on Weka 109
6.15 Comparison of TAF to the most competitive methods regarding training time of f. 110
6.16 Results of the two-tailey Nemenyi test on SVM. 112
6.17 Results of the two-tailey Nemenyi test on Weka. 112
6.18 Results of the Bayesian Hierarchical test for TAF against all other competitors on SVM. 113
6.19 Results of the Bayesian Hierarchical test for TAF against all other competitors on Weka. 115
6.20 Runtime comparison of SGPT, FMLP and full GP models. 116

7.1 Average Rank across all competing methods on Weka 121
7.2 Average Normalized Accuracy of each competitor on Weka 122
7.3 Overview of how many times FMLP picks which model 124
7.4 Average Rank of all competitors including all tasks 125
7.5 Average Hyperparameter Rank of all competitors including all tasks. 126
7.6 Average hyperparameter rank for binary and multiclass classification tasks . 127
7.7 Average Hyperparameter Rank for the joint classification task (left) and for regression (right) 128

A.1 A multivariate Gaussian defined on two variables, the z-axis shows the density of the variable $x = (x_1, x_2)$ 136

CHAPTER 1

INTRODUCTION

In the modern times, making sense of large amounts of gathered data is key to making advancements in many research areas such as medicine and natural sciences, but also in economical areas like e-commerce, smart production in the industry 4.0 era and many others. As the amounts of data are too large and their dimensionality usually is too high, humans are generally overwhelmed when dealing with the data in a manual fashion. Therefore, a lot of research in machine learning is conducted on understanding the data with the aid of machines, i.e. computers, and making use of it with models that are able to learn from data to then predict variables of interest for the future. Applications for machine learning range from predicting health conditions of humans over automatically detecting objects in a camera image for autonomous driving to webshops that want to maximize profit by showing the user a customized shop where items of interest to this user are preferred. In supervised learning - which will be the machine learning area that will be adressed in this thesis - the goal is usually to learn accurate prediction models from data where one has already observed the variable of interest, coining the name *supervised* learning. For a given input, for example some customer data and product features, these prediction models are able to predict a certain output, such as whether the customer has interest in the product.

Most commonly, the prediction models contain a lot of parameters, which are fitted to reproduce the observed data in the most meaningful way. Depending on the given task, i.e. whether it is classification, regression or ranking, the learning process is usually done by defining an objective function and optimizing it with respect to the model parameters.

However, not all of these parameters can be learned from the data using a well-defined ojective function, for example parameters that define how a model is optimized. Another example are parameters that define the complexity of the prediction model. From now on, we will call these parameters *hyperparameters* to differentiate them from the model parameters that can be learned from data. Hyperparameters usually have to be set manually prior to starting the optimization procedure, additionally, finding the optimal hyperparameters for the given application is usually very challenging, due to the fact that the hyperparameter spaces - the sets from which we can choose hyperparameters - do not have a finite cardinality. Consequently, many researchers and machine learning users rely on either hand-tuning these parameters or searching them exhaustively, which is largely time-consuming and in general produces a lot of useless computations that are simply being discarded after optimal hyperparameters have been found.

This thesis approaches the problem of hyperparameter optimization in supervised machine learning. Under the term hyperparameter optimization we also subsume model choice as the selection of a model to be learned effectively can be understood as setting another hyperparameter in a hierachy above the model-based hyperparameters. The primary goal of this thesis is therefore to develop methods that perform an intelligent hyperparameter optimization that work autonomously and thus aid in facilitating the overall process. In the following section we give a short overview of the structure of the thesis, as well as the main contributions that will be presented.

1.1 Overview

The overall structure of the thesis is given as follows: The next chapter *Hyperparameter Optimization in Machine Learning* starts out by firstly defining supervised machine learning problems as the problem of empirical risk minimization. Thereafter, we will give a definition of the problem of hyperparameter optimization for a very general machine learning algorithm, which also considers model choice, learning algorithm, model structure and preprocessing of the data for instance. As we will see, the hyperparameter optimization problem turns out to optimizing a black-box function whose inputs are the hyperparameters. In the subsequent chapter *Bayesian Hyperparameter Optimization* we will introduce sequential model-based optimization which is a technique commonly used in black-box optimization and therefore is the basis of hyperparameter optimization methods. Additionally this chapter identifies, groups and presents related work for the task of hyperparameter optimization and discusses the competitor methods that will be

used for the experiments. The chapter *Experimental Setup* details on the creation of the different meta-data sets that have been used for empirical evaluation and defines the evaluation measures which are employed to assess the performance of a hyperparameter optimization method. The following three chapters introduce the main contributions of this thesis, where firstly the chapter *Relational Hyperparameter Optimization with Factorized Multilayer Perceptrons* presents a surrogate model that uses a combination of multilayer perceptrons and factorization models. This yields a surrogate that is capable of combining a high level of abstraction with learning latent representations for binary indicator variables as many models in the area of recommender systems are doing.

The following chapter *Scalable Hyperparameter Optimization with Gaussian Process Ensembles* introduces surrogate models that are based on a combination of Gaussian processes, more specifically an ensemble of Gaussian processes, which makes them probabilistic and accurate, while maintaining scalability properties that a joint Gaussian process is not able to offer anymore. In this chapter we will also present a slight alternative to using an ensemble of Gaussian processes as surrogate model, by introducing a transfer acquisition function, where the individual GPs are weighted in the Expected Improvement acquisition function, in opposition to their scores being assembled before being evaluated by the acquisition function.

The subsequent chapter *Hyperparameter Optimization Across Problem Instances* is concerned with the problems of learning hyperparameter performance across different problem instances such as different models and associated learning algorithms. Additionally, also the problem of learning the same models for different problem tasks, i.e. regression and classification is being investigated. In this chapter, a special focus is being laid on the performance of the surrogate models that have been proposed in the previous chapters.

Finally, the thesis will be summarized by the *Conclusion* chapter, where firstly the methods proposed for hyperparameter optimization in this thesis are being reviewed. Secondly, directions for future work in autonomous machine learning especially hyperparameter optimization are discussed.

1.2 Main Contributions

The main goal of this thesis is to present research that was conducted on the task of hyperparameter optimization, the contributions are manifold and can be summarized as:

- **Formalization of the state of the art and the proposed surrogate models**

using a single notational framework. We formalize the problem of hyperparameter optimization as a black-box optimization problem and show how Bayesian optimization can be used to steer hyperparameter search methods. In addition to that, we present the state of the art using a single notational framework, allowing the reader to cast a glance at all different methods without being overwhelmed by different notation schemes.

- **Proposition of using surrogate models that are able to learn latent embeddings of the hyperparameter space.** We introduce Factorized Multilayer Perceptrons, which are a combination of feedforward neural networks and therefore allow to map hyperparameters to latent features, this can be used to learn dataset characteristics from an indicator variable.

- **Proposition of scalable surrogates that are based on Gaussian process ensembles.** We introduce a variety of transfer surrogate models which are essentially ensembles of Gaussian processes learned on the hyperparameter performance of each individual data set, thus are able to scale up to large amounts of meta data, while maintaining the useful properties of Gaussian processes.

- **Hyperparameter optimization across problem instances.** We not only learn hyperparameter performance across data sets, but also generalize across different problem tasks, i.e. regression and classification problems, as well as different model choices.

- **Experimental evaluation & data sets for hyperparameter optimization.** For all of the problems and their corresponding solutions addressed in this thesis, we perform a detailed experimental comparison on meta data sets that were created by ourselves. We describe in detail how the meta data sets were built, including the hyperparameter choice, the choice of data sets, the software that was used and the like. In addition, all meta data sets are publicly available to facilitate further experiments in future work.

1.3 Published Work

The work presented in this thesis has been published on international machine learning and data mining conferences and in the machine learning journal. The publications that we will present in the context of Bayesian hyperparameter optimization are:

- **Scalable Gaussian process-based transfer surrogates for hyperparameter optimization**
  N Schilling, M Wistuba, L Schmidt-Thieme
  Machine Learning, 1-36 (2017)

- **Scalable Hyperparameter Optimization with Products of Gaussian Process Experts**
  N Schilling, M Wistuba, L Schmidt-Thieme
  Joint European Conference on Machine Learning and Knowledge Discovery in Databases (2016)

- **Joint Model Choice and Hyperparameter Optimization with Factorized Multilayer Perceptrons**
  N Schilling, M Wistuba, L Drumond, L Schmidt-Thieme
  IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), 2015

- **Hyperparameter optimization with Factorized Multilayer Perceptrons**
  N Schilling, M Wistuba, L Drumond, L Schmidt-Thieme
  Joint European Conference on Machine Learning and Knowledge Discovery in Databases (2015)

- **Hyperparameter Optimization Across Problem Tasks**
  N Schilling, T Windler, L Schmidt-Thieme
  Archives of Data Science (currently being reviewed) (2017)

However, while I was pursuing my doctorial studies at the Information Systems and Machine Learning Lab at the University of Hildesheim, I have co-authored several publications that will be listed below:

- **Towards Distributed Pairwise Ranking using Implicit Feedback**
  M Jameel, N Schilling, L Schmidt-Thieme
  in The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2018)

- **Automatic Frankensteining: Creating Complex Ensembles Autonomously**
  M Wistuba, N Schilling, L Schmidt-Thieme
  Proceedings of the 2017 SIAM International Conference on Data Mining, 741-749 (2017)

- **Near real-time Geolocation Prediction in Twitter Streams via Matrix Factorization based Regression**
  N Duong-Trung, N Schilling, L Schmidt-Thieme
  Proceedings of the 25th ACM international on conference on information and knowledge management (2016)

- **Hyperparameter Optimization Machines**
  M Wistuba, N Schilling, L Schmidt-Thieme
  IEEE International Conference on Data Science and Advanced Analytics (DSAA) (2016)

- **Calculation of upper esophageal sphincter restitution time from high resolution manometry data using machine learning**
  M Jungheim, A Busche, S Miller, N Schilling, L Schmidt-Thieme, M Ptok
  Physiology & Behavior 165, 413-424 (2016)

- **Bank Card Usage Prediction Exploiting Geolocation Information**
  M Wistuba, N Duong-Trung, N Schilling, L Schmidt-Thieme
  arXiv preprint arXiv:1610.03996 (2016)

- **Two-Stage Transfer Surrogate Model for Automatic Hyperparameter Optimization**
  M Wistuba, N Schilling, L Schmidt-Thieme
  Joint European Conference on Machine Learning and Knowledge Discovery in Databases (2016)

- **Latent Time-Series Motifs**
  J Grabocka, N Schilling, L Schmidt-Thieme
  ACM Transactions on Knowledge Discovery from Data (TKDD) 11 (1), 6 (2016)

- **Learning DTW-shapelets for Time-Series Classification**
  M Shah, J Grabocka, N Schilling, M Wistuba, L Schmidt-Thieme
  Proceedings of the 3rd IKDD Conference on Data Science, 2016, 3

- **Sequential Model-Free Hyperparameter Tuning**
  M Wistuba, N Schilling, L Schmidt-Thieme
  IEEE International Conference on Data Mining (ICDM), 2015 1033-1038

- **Learning Hyperparameter Optimization Initializations**
  M Wistuba, N Schilling, L Schmidt-Thieme

IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2015.

- **Learning Data Set Similarities for Hyperparameter Optimization Initializations**
  M Wistuba, N Schilling, L Schmidt-Thieme
  MetaSel@ PKDD/ECML (2015), 15-26

- **Hyperparameter Search Space Pruning,Äìa new Component for Sequential Model-Based Hyperparameter Optimization**
  M Wistuba, N Schilling, L Schmidt-Thieme
  Joint European Conference on Machine Learning and Knowledge Discovery in Databases (2015)

- **Event Prediction in Pharyngeal High-Resolution Manometry**
  N Schilling, A Busche, S Miller, M Jungheim, M Ptok, L Schmidt-Thieme
  Data Science, Learning by Latent Structures, and Knowledge Discovery, 341-352 (2015)

- **Learning Time-Series Shapelets**
  J Grabocka, N Schilling, M Wistuba, L Schmidt-Thieme
  Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (2014)

CHAPTER 2

HYPERPARAMETER OPTIMIZATION IN MACHINE LEARNING

Before we are able to formally define the problem of hyperparameter optimization, we shortly introduce the main problem in supervised machine learning, which is actually learning a model through empirical risk minimization. Additionally, we attempt to group hyperparameters into four different classes, which are related to the choice of objective function, the learning algorithm, the model itself and finally the data processing. We will identify plenty of examples for all these groups. Finally, we define the hyperparameter optimization problem in the sense of empirical risk minimization for a single data set, as well as for the scenario where hyperparameter performance has already been observed on past data sets and problem settings, where the inclusion of such observations is supposed to be fruitful.

2.1 Supervised Machine Learning

The ultimate goal of supervised machine learning is to estimate a model that for a given input is able to predict a certain output, which is of one's interest. This very general framework applies to many real world applications for example in medicine, where the interest is to predict the state of health of a patient, in e-commerce, where the goal is to predict whether a customer is interested in a certain product and many more. Let us formally denote the input by $x \in \mathcal{X}$ where $\mathcal{X}$ is the space of all possible inputs. Usually,

the inputs are called features, a term that will be frequently used throughout this thesis. By $y \in \mathcal{Y}$, we denote the output and its respective space, the output is usually called a ground-truth, target or label, if it is observed. Depending on the task, the output space differs, consider for example $\mathcal{Y} = \mathbb{R}$ for a regression task and $\mathcal{Y} = \{0, 1\}$ for a binary classification problem.

It is assumed that the collected data was generated by an unknown probability distribution $\mathcal{P}$ over the input and output space,

$$\mathcal{P} : \mathcal{X} \times \mathcal{Y} \longrightarrow \mathbb{R}_0^+ \tag{2.1}$$

and the goal is to predict targets y given some input x. Thus we aim to learn a model

$$m : \mathcal{X} \longrightarrow \mathcal{Y} \tag{2.2}$$

which predicts the labels most accurately. Such model is found by minimizing a risk functional

$$\text{risk}(m, \mathcal{P}) = \mathbb{E}_{(x,y)\sim\mathcal{P}}(\ell(y, m(x))) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(y, m(x)) \, \mathrm{d}\mathcal{P}(x, y) \tag{2.3}$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}_0^+$ is a loss function that evaluates how similar the prediction and the label are, for a regression task a common choice is the least squares loss

$$\ell(y, m(x)) = (y - m(x))^2 \ . \tag{2.4}$$

As the data generating distribution $\mathcal{P}$ is unknown, the integral in equation 2.3 is approximated by evaluating it only on a finite set of data points that have been observed. Hence, the risk is approximated by the empirical risk

$$\widehat{\text{risk}}(m, D) = \frac{1}{|D|} \sum_{(x,y)\in D} \ell(y, m(x)) \tag{2.5}$$

where D is a (training) data set that was generated by $\mathcal{P}$. Usually, the prediction models m are parametrized by θ, i.e.

$$m(x) = m(x; \theta) \ . \tag{2.6}$$

The optimal parameters $\theta^\star$ are then found by minimizing the empirical risk

$$\theta^\star = \arg\min_\theta \widehat{\text{risk}}(m(\theta), D) \ . \tag{2.7}$$

However, minimizing solely the empirical risk as in the equation above yields a model that overfits the training data. This results in poor generalization performance when the

model is queried for unseen data, which is the prime cause for machine learning. For many problems it suffices to put a Gaussian prior with zero mean on the parameters θ, which in the maximum a posteriori estimation results in the minimization of the Tikhonov regularization term

$$\text{reg}(\theta, \lambda) = \lambda \|\theta\|_2^2 \ . \tag{2.8}$$

Here, $\lambda \in \mathbb{R}^+$ is a hyperparameter that is inversely proportional to the variance of the Gaussian prior on θ. It therefore determines the strength of regularization that is put on the model parameters. Overall the minimization problem becomes

$$\theta^\star = \arg\min_\theta \widehat{\text{risk}}(m(\theta), D) + \text{reg}(\theta, \lambda) \ . \tag{2.9}$$

Setting the correct amount of regularization in order to learn a model that fits the training data decently while still offering good generalization performance is generally difficult as λ can not be learned from the data. If λ is set too small, the model likely will still overfit to the signal of the training data, if λ is set too high the model will become random. Obviously, a straightforward optimization for the above problem yields that the optimal λ is $\lambda = 0$, which totally undermines the whole purpose of using a regularization term. Because of this, many researchers rely on testing several different configurations for λ, and in the end choose that configuration that delivers the smallest error on validation data which is distinct from the training data. Optimizing the amount of regularization is one of the many instances of the hyperparameter optimization problem, which will be detailed in the next section.

2.2 Occurences of Hyperparameters in Machine Learning

Before we introduce the hyperparameter optimization problem in general, it is necessary to discuss all kinds of different hyperparameters that appear in machine learning. In the previous section we have seen that finding the correct setting of the regularization coefficient λ in empirical risk minimization is one instance of a hyperparameter that needs to be optimized. In the following subsections, different instances of hyperparameters will be presented.

2.2.1 Objective Function

Many hyperparameters are concerned with the objective function that is optimized, which usually consists of a loss term over the training data and a regularization term to prevent the learned model from overfitting. Choosing the loss function is a hyperparameter that determines the solution, in a regression task, choosing the least squares loss instead of the absolute error might yield a model that is overadapted to the outliers, as the error gets squared. For classification problems, one may choose between losses such as the Hinge loss [74], the squared Hinge loss and the logistic loss. The same choice exists for the regularization term, as for example in equation 2.8, where a Tikhonov regularization term [90] was used. Tikhonov regularization is based on the L2 metric, however, one may also choose the L1 metric, which induces sparsity on the model parameters [89]. Additional regularization terms based on the graph Laplacian of the data set can be employed to encourage the learned model the be smooth, i.e. for similar inputs deliver similar outputs, this technique is usually used in semi-supervised learning problems as for example in manifold regularization [5].

2.2.2 Learning Algorithm

Another set of hyperparameters to optimize is concerned with the learning algorithm, which is used to estimate well-performing model parameters throughout the training procedure. In many machine learning applications, first-order optimization methods such as stochastic gradient descent are most widely used [24]. Defining the correct step size, i.e. the length that one follows the gradient to update the model parameters is a crucial choice [76]. If it is set too high, the models usually diverge, as they tend to jump over the minimum, which is why small step sizes are usually preferable, although they may induce slow convergence. Out of these reasons, adaptive step sizes such as a cooldown schedule, the bold-drivers scheme that slightly increases the step size if the loss was decreased and drastically decreases it if the error increases, have been used. More recently, using adaptive step sizes individually for each model parameter is accomplished using the famous Adagrad scheme [21] which was extended to Adadelta [97]. In conclusion, a parameter such as the step size crucially influences the convergence of the learning algorithm and the choice how to deal with it certainly is a hyperparameter.

Additionally, for some problems second-order quasi Newton methods are used such as the famous L-BFGS algorithm [54], which incorporates the curvature of the loss function into the parameter updates. Since the introduction of stochastic gradient descent by

Leon Bottou [9], choosing the number of instances to consider for computing the updates is a hyperparameter that needs to be set correctly. Usually, it is set to one instance, but in some areas such as image classification and segmentation, higher batch sizes are employed.

Besides gradient based learning algorithms, the user may choose to utilize sampling-based learning algorithms such as Markov Chain Monte Carlo [1] or hybrid monte carlo [65]. These learning algorithms usually sample model parameters in the vicinity of an earlier estimate and accept them if they are working better, creating a Markov chain of model parameters. For MCMC to work, the proposal distribution that samples in the vicinity needs to be set up properly, otherwise it may lead to poor convergence.

Overall, there are many hyperparameters one can consider when it comes to choosing the right learning algorithm as well as the right parametrization for the problem at hand.

2.2.3 Model Choice

When it comes to model choice, the first obvious hyperparameter is the model choice itself, for instance choosing a neural network over a linear regression or a polynomial regression. The model choice usually influences the final prediction performance as well as the learning time quite heavily, as more complex models are usually more powerful, but also require more time to converge. Choosing the right model depends on the task itself and the resources that one wants to commit.

All other hyperparameters that are concerned with the model choice are usually complexity parameters. In a factorization model such as [69], the dimensionality of the latent feature space has to be set accordingly. For neural networks, the structure has to be defined by setting how many layers are supposed to be used, how many neurons are used or which activation function is used. All these parameter significantly influence the structure and complexity of the network. When learning a support vector machine, the choice of kernel function, i.e. linear kernels, Gaussian kernels or polynomial kernels can also be understood as a hyperparameter as it highly influences the results [84]. Finally, for a tree based model such as [12] the arity of splits, the depth of the tree and the minimum leaf size are hyperparameters that also determine the complexity of the model.

2.2.4 Data Processing

Lastly, the choice how to handle the data before starting the learning algorithm is another class of hyperparameters. In many applications, whether it is supervised or unsupervised learning, the data is normalized as many models are not able to deal with features having different scales [43] [23]. One way to accomplish this is to scale the data to be standard Gaussian, this usually works quite well; however, if we are facing highly sparse data this preprocessing makes the data dense which is not in our interest.

For many other domains besides sparse data, using a Gaussian standardization is not sufficient, as the data is not given in vectorial form but has richer structure such as images, or has a sequential aspect such as time-series. In these areas, more sophisticated chains of preprocessing steps are implemented. For the problem of speaker identification on raw audio data, the work by [60] first uses Hamming windows and zero padding on the raw data, to then compute a short term Fourier transformation. On this data, filter-bank energy features are computed which are then decorrelated and standardized afterwards. In many areas, heavy preprocessing routines such as the one above are used, which have a tremendous influence on the performance of the learning system that we end up with.

2.3 The Hyperparameter Optimization Problem

In this section, we will describe the hyperparameter optimization problem in two scenarios, once where meta knowledge is not present and once where we aim to exploit it.

2.3.1 Hyperparameter Optimization without Meta Knowledge

Following the notation similar to [7] let us denote the space of all data sets by $\mathcal{D}$. It consists of all possible data sets, instances might have a vectorial representation, but can also have different structures such as sequences or images. Additionally, let $\mathcal{M}$ be the space of all machine learning models, including parametric models such as linear and logistic regression and neural networks, as well as non parametric models such as decision trees and nearest neighbor models. Moreover, let Λ denote the hyperparameter configuration space, in this way $\lambda \in \Lambda$ is a hyperparameter configuration that determines all of the above mentioned areas such as the choice of objective function, learning algorithm and the like. The configuration space Λ can be seen as infinitely dimensional,

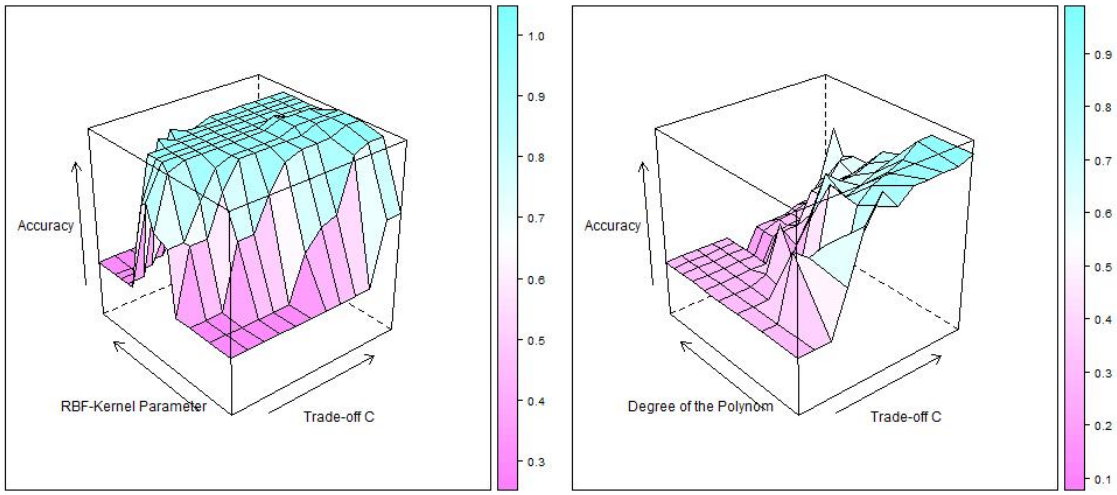

Figure 2.1: Response surfaces of learning a support vector machine on the famous Iris data set. The left plot shows the validation accuracy for an SVM with RBF kernel, kernel width and cost of the slack variables. The right plot shows the responses of an SVM with a polynomial kernel, for varying kernel degrees and slack variable cost.

as new models, learning algorithms and data processing strategies are developed on a daily basis. However, when it comes to experiments, Λ is usually treated as a finite dimensional space, simply by restricting the set of model and learning algorithms to a discrete subset.

Finally, let us define $\mathcal{A}$ as a most general machine learning algorithm, that takes as input a hyperparameter configuration λ and a data set D and returns a model m that was learned on the training partition D^{train} of the data set D.

$$\mathcal{A} : \Lambda \times \mathcal{D} \longrightarrow \mathcal{M} \tag{2.10}$$

Given a configuration λ, $\mathcal{A}$ searches through $\mathcal{M}$ to find a model that minimizes the objective function

$$\mathcal{O} : \Lambda \times \mathcal{M} \times \mathcal{D} \longrightarrow \mathbb{R}_0^+ \ . \tag{2.11}$$

In most cases, the objective function consists of a loss term and a regularization term as was presented in the previous section. Consequently, we can define $\mathcal{A}$ as

$$\mathcal{A}(\lambda, D) = \arg\min_{m \in \mathcal{M}} \mathcal{O}(\lambda, m, D^{\text{train}}) \ . \tag{2.12}$$

When evaluating the quality of the learned model given a hyperparameter configuration λ, many researches rely on the cross-validation scheme. In K-fold cross validation,

the training data is split into K many equally sized parts. Then, given a hyperparameter configuration, the model is learned K many times where each data instance appears in the validation partition once, and in the training partition $K-1$ many times. Afterwards, the empirical risk on the respective validation partitions is averaged and used to determine the optimal hyperparameters. In this way, using the cross validation score allows for defining the problem of hyperparameter optimization. The optimal hyperparameter configuration $\lambda^\star$ is the hyperparameter configuration $\lambda \in \Lambda$ which minimizes the cross validation score

$$\lambda^\star = \arg\min_{\lambda \in \Lambda} \frac{1}{K} \sum_{k=1}^{K} \widehat{\text{risk}}\left(\mathcal{A}(\lambda, D_k^{\text{train}}), D_k^{\text{valid}}\right) \tag{2.13}$$

where D_k^{train} and D_k^{valid} are the respective training and validation partitions to learn and evaluate the model. To tighten the notation, we will denote the cross validation score as a function f

$$f(\lambda, D) = \frac{1}{K} \sum_{k=1}^{K} \widehat{\text{risk}}\left(\mathcal{A}(\lambda, D_k^{\text{train}}), D_k^{\text{valid}}\right) \tag{2.14}$$

where the contours of f for a fixed data set and varying λ is also called response surface. Figure 2.1 shows an exemplary response surface of a simple classification problem of learning a support vector machine on the well-known Iris data set. From the small example, it is already visible that these response surfaces are highly nonlinear and in some cases discontinuous as well.

2.3.2 Hyperparameter Optimization Across Data Sets

Usually, when machine learners optimize their hyperparameters for a familiar model that is to be learned for new data, they already have a little insight into how the hyperparameter choices influence the final performance. This knowledge is usually gained from past experiments with the same or at least a similar model. In recommender systems for instance, the dimensionality of the latent representation usually shows a saturation curve if the model is properly regularized. This means that it is crucial to have at least a few latent features, but going beyond a certain dimensionality typically does not improve the model's performance anymore before it starts to overfit [75]. With such knowledge, the infinite configuration space might become more manageable.

Out of these reasons, much research has been devoted to including hyperparameter performance that has been observed on past data sets in order to speed up the hyperparameter optimization process for the new data set. Let us assume we have optimized

hyperparameters for N data sets already, meaning we at least have gathered some observations of hyperparameter performance on the data sets $D_1,...,D_N$. By $\mathcal{H}$, we denote the observation history, which is then the set of all past observations

$$\mathcal{H} = \bigcup_{\lambda \in \Lambda^1} f(\lambda, D_1) \cup \ldots \cup \bigcup_{\lambda \in \Lambda^N} f(\lambda, D_N) \tag{2.15}$$

where Λ^i is the discrete subset of Λ for which f has been observed on data set D_i as the employed hyperparameter configurations usually differ from one data set to another. Throughout this thesis, we also use the term meta-knowledge for the observation history, to account for the meta-learning background. Finally, the problem of hyperparameter optimization across data sets is given an observation history $\mathcal{H} \subset \Lambda \times \mathbb{R}$ to find that configuration $\lambda^\star \in \Lambda$ which minimizes the cross validation loss, as in Equation 2.13, while making efficient use of $\mathcal{H}$, for example by proposing to test hyperparameters that are more likely to perform well.

2.4 A Word on Notation

In the scenario above, we have introduced machine learning as the task of learning a model m on some data pairs (x, y), where the task is to predict the measured output y as accurate as we possibly can. Learning these models is conditioned on hyperparameters λ, which greatly influence the model that we learn, mainly in terms of accuracy and generalisability. In the upcoming sections, we will develop methods that put machine learning over machine learning, i.e. we try to learn models that are able of predicting the response surface, given the hyperparameters λ. As now the hyperparameters are the input to the models, we will from now on start by denoting them using x, which feels more natural than using λ, since the expression $f(x, D)$ is more intuitive over $f(\lambda, D)$. Following this approach, from now on, when we discuss some $x \in \mathcal{X}$, we usually mean a vector of hyperparameters λ that serves as input to our models which try to learn the response surface. There is one exception to this rule, which is chapter four, where the experimental setup is described, and both data features and hyperparameters are used.

Additionally, we will drop the dependence of f on the data set D whenever it is not needed in the current context, in order to keep the notation as expressive but also as simple as possible.

CHAPTER 3

BAYESIAN HYPERPARAMETER OPTIMIZATION

In the last chapter, we have seen the problem of hyperparameter optimization for supervised machine learning problems. In this chapter we will present sequential model-based optimization (SMBO) which is a black-box optimization strategy that perfectly suits the problem of hyperparameter optimization. Afterwards, we will introduce and discuss the most recent related work on hyperparameter optimization, both those methods that are based on SMBO and those that are not.

3.1 Bayesian Optimization

Before we discuss sequential model-based optimization, we will introduce the general framework of Bayesian optimization, of which sequential model-based optimization then is a special case. In general, Bayesian optimization [62] attempts to find the global minimizer $x^\star$ (or maximizer) of an unknown function f:

$$x^\star = \arg\min_{x \in \mathcal{X}} f(x) \tag{3.1}$$

where the search is usually realised in a compact subset of $\mathcal{X}$. The function f is also called a black-box function as there is no closed form given. Additionally, there are no properties such as convexity of the function f assumed, which is used a lot in convex optimization theory. However, f can be queried for an arbitrary configuration $x \in \mathcal{X}$ and will deliver a noisy observation $y(x)$

$$y(x) = f(x) + \epsilon_x \tag{3.2}$$

where ϵ_x is a random variable that explains the noise in the measurements. In many scenarios, the noise is either set to zero, leading to $y = f$, or is modeled in a simple manner. If random effects are modeled, the noise is usually considered to be homoscedastic, which means that the level of noise is the same everywhere in the configuration space. One common choice is for example Gaussian noise

$$\epsilon_x \sim \mathcal{N}(0, \sigma^2) \ . \tag{3.3}$$

The goal of Bayesian optimization is to find the minimizer $x^\star$ of the true underlying function f, this is accomplished by a steered search that sequentially chooses each new point to evaluate while considering all of the data that has been observed so far. Thus, the output of the n-th decision is defined as

$$x_n = d_n(x_1, ..., x_{n-1}, y_1, ..., y_{n-1}) = d_n(\mathcal{H}) \tag{3.4}$$

where $y_i = y(x_i)$ and we introduce the observation history $\mathcal{H}$ to shorten the notation a bit. Please note that in contrast to Equation 2.15, we now assume the measurements to be noisy, therefore $\mathcal{H}$ contains measurements denoted as y instead of f.

In contrast to standard optimization, Bayesian optimization always considers the whole data, as opposed to optimization algorithms such as gradient descent for a parametric model that generally only considers first order information in the vicinity of the last point, which might be noisy but is usually very fast to compute. Bayesian optimization, however, considers all the data hence also has a more complex proposal calculation.

At this point, we can define a utility function $u(f, d)$, that measures the utility of decision d. Then, we can simply define the optimal decision as the one that maximizes the utility

$$d^\star = \arg\max_d u(f, d) \ . \tag{3.5}$$

However, solving this optimization problem requires full knowledge of f, which we do not have. Bayesian optimization treats f as a variable out of a family of functions $f \in \mathcal{F}$ that are all possible solutions and puts a prior $p(f)$ on it. For example, for all real valued linear functions of the form $f(x; \theta) = \theta^\top x$ with $\theta \in \mathbb{R}^N$, we could assume that the average function is zero, i.e. use multivariate Gaussian prior

$$\theta \sim \mathcal{N}(0, \Sigma) \tag{3.6}$$

where Σ is a covariance matrix of the parameters. Having defined such a prior distribu-

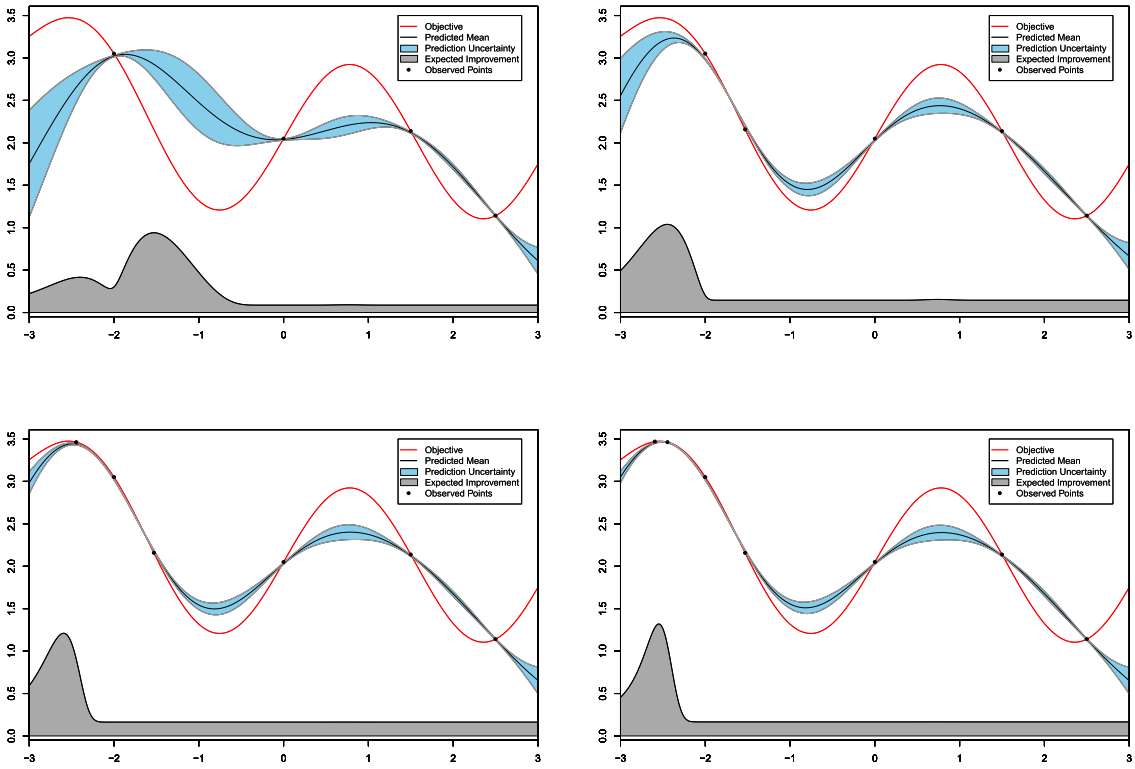

Figure 3.1: Overview of how Bayesian optimization works in practice when maximizing a one-dimensional function. Before Bayesian optimization starts we have observed the function (red line) at four points. The posterior mean using a Gaussian process prior as well as its uncertainty is shown in blue. The decision on the next point is made by the maximal expected improvement which is shown on the bottom in gray. After the proposed value has been observed the posterior adjusts to the new data points, and a new point is chosen, where in the final two iterations the surrogate is exploited and the maximum is found.

tion on f, we can compute the optimal decision $d^\star$ as the decision that maximizes the utility in expectation over $\mathcal{F}$

$$d^\star = \arg\max_d \int_{\mathcal{F}} u(f,d)\,\mathrm{d}p(f)\ . \tag{3.7}$$

However, using the above equation to estimate the optimal next decision does not include the previously observed data, as we are only integrating with respect to the prior

distribution of f. Using Bayes' rule we can write the posterior

$$p(f \mid \mathcal{H}) = \frac{p(\mathcal{H} \mid f)\, p(f)}{p(\mathcal{H})} \propto p(\mathcal{H} \mid f)\, p(f) \tag{3.8}$$

where the term in the denominator is simply the probability of observing the data which does not depend on f and therefore can also be left out. Using the posterior, we finally arrive at

$$d^\star = \arg\max_d \int_{\mathcal{F}} u(f, d)\, \mathrm{d}p(f \mid \mathcal{H}) \tag{3.9}$$

which is the key equation for Bayesian optimization. However, we need to ensure that the integral on the right side of the above equation is analytically tractable, and can be computed in much less time than solving our initial optimization problem. A common choice for prior and likelihood is to use conjugated priors, where the posterior then has a known distribution. For the choice of a Gaussian process prior over functions and the improvement as utility function, an optimization sequence can be seen in Figure 3.1 where a non-convex one-dimensional function is optimized.

To conclude, Bayesian optimization requires the choice of two things, at first a prior $p(f)$ over functions that covers our general belief of the domain we are operating in but still manages to be analytically computable, secondly a way of finding the next decision. The choice of the prior and the likelihood of the observations will give rise to the posterior that we need to compute the optimal Bayesian decision. We will subsume both in a so called surrogate model that we will introduce in the next section. In order to find the next decision, Bayesian optimization employs a utility function $u(f, d)$ which has to be specified before starting sequential model-based optimization. We will see in the next section how different choices of utility functions lead to different decisions being made, this will be comprised in the so-called acquisition function.

3.2 Sequential Model-Based Optimization

Sequential model-based optimization is a technique that is used for optimizing black-box functions [42]. The cross-validation performance of a learned model given a hyperparameter configuration as given in equation 2.14 certainly is a black-box, considering the interaction of hyperparameter configurations on the performance of a learned model. Moreover, fitting a model to training data and applying it to validation data in order

to measure its generalization capabilities takes a lot of time and is therefore costly to assess. These reasons make sequential model-based optimization fit perfectly for the task of hyperparameter optimization.

As the black box can only be queried, the goal in sequential model-based optimization is to learn a so-called surrogate model Ψ that for a given configuration x predicts a distribution over possible outputs $f(x)$. The main property of the surrogate is that it has to be cheap to evaluate, if we would use a surrogate model where inference is costly, we could evaluate the response f directly instead. Throughout this thesis we will always assume that our surrogate Ψ is a random variable that follows a Gaussian distribution

$$\Psi(x) = p(f(x) \mid \mathcal{H}) = \mathcal{N}\left(f(x) \mid \mu(f(x)), \sigma^2(f(x))\right) \quad . \tag{3.10}$$

It takes as input the hyperparameter configuration and predicts a mean $\mu(x)$ and a variance $\sigma^2(x)$ of the validation loss that we likely are to see. From now on, we will assume that we are querying Ψ for a specific configuration $x_\star$, which we are interested in. Therefore, we will denote $f_\star = f(x_\star)$ and also use $\mu_\star = \mu(f_\star)$ and $\sigma^2_\star = \sigma^2(f_\star)$. This overall shortens the notation of a surrogate to

$$\Psi(x_\star) = \mathcal{N}\left(f_\star \mid \mu_\star, \sigma^2_\star\right) \quad . \tag{3.11}$$

As we have seen above, Bayesian optimization puts a prior over all possible surrogate models $p(\Psi)$, which covers a general belief of a model's properties in a certain domain, for example continuity or smoothness of the outputs of the function. After having gathered observations of the unknown function in $\mathcal{H}$, the prior can be combined with the likelihood of the observations determining how well a model fits these observations. The resulting posterior is a distribution over models, for predictions one usually applies the posterior mean, however, the shape of the distribution is also very useful for uncertainty estimation, a component that will be a requirement for sequential model-based optimization, as we have to integrate over all possible outcomes.

As the name suggests, sequential model-based optimization works sequentially. After having observed the black box for some time a surrogate model is fitted to the observation history. Since the surrogate model Ψ can be queried easily, its predictions are used to determine which configuration to test next, this is usually done by maximizing an acquisition function

$$a : \mathcal{X} \longrightarrow \mathbb{R}^+ \tag{3.12}$$

that takes as input an unobserved configuration to predict a score for it. The acquisition function uses the predicted mean, the predicted variance of the surrogate model as well

Algorithm 1 Sequential Model-based Optimization

Require: Hyperparameter configuration space $\mathcal{X}$, observation history $\mathcal{H}$, number of trials T, acquisition function a, surrogate model Ψ.
Ensure: Best hyperparameter configuration found.

$f^{\min} \leftarrow \infty$
for $t = 1$ to T **do**
    Compute $\Psi(x) = p(f(x)|\mathcal{H})$
    $x \leftarrow \arg\max_{x \in \mathcal{X}} a(\Psi(x))$
    Evaluate $f(x)$
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(x, f(x))\}$
    if $f(x) < f^{\min}$ **then**
        $x^{\min}, f^{\min} \leftarrow x, f(x)$
return $x^{\min}$

as the currently best response value in order determine the scoring. The hyperparameter configuration with the highest score is then used to evaluate the black box function f, which is the most costly step. Afterwards, the observation is added to the observation history and the surrogate model is relearned. The whole process is repeated until either a certain number of trials has passed or a well-performing configuration is found. An overview of the whole process can be seen in Algorithm 1.

SMBO is a very general approach that works for any hyperparameter optimization problem and not only for specific problems. By searching through $\mathcal{X}$ in a controlled matter it also requires less evaluations of the response surface as opposed to exhaustive search methods such as grid-search. However, the main disadvantage is that sequential model-based optimization is sequential in its nature as the surrogate model is relearned after every black box evaluation to propose the next configuration.

After having learned a surrogate model on some observations, the task of the acquisition function is to decide which configuration should be tested subsequently, while keeping a reasonable tradeoff between exploitation and exploration by taking into account both the surrogate's predictions and uncertainties.

Initially, we have only tested a few hyperparameter configurations, making the observed response surface quite sparse. In such a situation, the main goal is probably to gather more information about the response surface which we call exploration. Since we are learning Ψ to reflect the past observations, its predictions are likely to be more accurate in the vicinity of the past configurations, consequently, the uncertainty increases in unobserved regions. When considering only uncertainty, it would be simple to always propose to test configurations with low certainty, simply to drive exploration and to gain

more knowledge of the response surface.

However, after a certain amount of trials has been conducted, the surrogate's predictions become more reliable and therefore should also be taken into account more when deciding which configuration to choose next. At this point we could simply only consider the configuration that has the highest predicted performance, completely ignoring the uncertainty of the surrogate. Assuming a continuous response surface, the new configurations will very likely lie in the vicinity of the currently best configuration. This process is called exploitation, as it simply exploits the knowledge of the surrogate model.

3.2.1 Acquisition Functions

It seems natural that in the beginning a larger portion of our efforts should go to exploration which then slowly changes over to exploitation in the meantime. To balance the tradeoff between exploration and exploitation, acquisition functions are used, we will discuss those presented in [30] in this subsection. Let u be a utility function that takes as input the predicted performance of an unobserved hyperparameter configuration $x_\star$ and measures its usefulness given the already observed hyperparameters. For example u could simply return one if the predicted performance is higher than the best observed performance so far and zero otherwise, effectively making every hyperparameter configuration useful that is expected to perform better. Using the utility function, we are able to define the acquisition function a as the expected utility

$$a(x_\star) = \int_{\mathcal{F}} u(x_\star)\, p(f_\star \mid \mathcal{H})\, \mathrm{d}f_\star \tag{3.13}$$

which is the same equation as Equation 3.9, where the Bayesian optimal next decision is being estimated. Choosing the utility function in different ways yields different acquisition functions if the integral can be solved analytically, or somehow estimated efficiently. In the following, we will give a short introduction into different utility functions and the acquisition function they provide.

Probability of Improvement

As was already mentioned before, a utility function could simply return a utility of one if the proposed hyperparameter configuration has a higher expected performance and zero otherwise, exactly this utility was proposed back in the days by Kushner [52].

Lemma 3.1. *For the utility function*

$$u(x_\star) = \begin{cases} 1 & if \quad \mu(x_\star) < f^{min} \\ 0 & otherwise \end{cases} \tag{3.14}$$

the acquisition can be computed as the probability of improvement

$$a^{PI}(x_\star) = \Phi\left(\frac{f^{min} - \mu(x_\star)}{\sigma(x)}\right) \quad . \tag{3.15}$$

Proof. If we plug this utility function into the definition of the acquisition function, we get

$$\begin{aligned} a^{\mathrm{PI}}(x_\star) &= \int_{\mathcal{F}} u(x_\star) p(f_\star \mid \mathcal{H}) \, \mathrm{d}f_\star \\ &= \int_{-\infty}^{f^{\min}} p(f_\star \mid \mathcal{H}) \, \mathrm{d}f_\star \\ &= \Phi\left(\frac{f^{\min} - \mu(x_\star)}{\sigma(x)}\right) \end{aligned} \tag{3.16}$$

where Φ is the cumulative distribution function of the standard Gaussian distribution. The first equality holds simply by the definition of our utility u and the second equality holds as $\Psi(x_\star)$ is Gaussian distributed, where the expression only needs to be standardized. ■

The downside of this fairly simple acquisition function is that it does exploitation of the surrogate only, as configurations close to already observed configurations will have a lower uncertainty. If these configurations then have a mean, that is only slightly better than $f^{\min}$, they will be chosen as next configuration to test. This downside can be alleviated by introducing a factor ξ of how much the configuration should be better, giving more chance for configurations with higher uncertainty. Donald Jones [41] describes the influence of ξ by saying that the method "is extremely sensitive to the choice of the target. If the desired improvement is too small, the search will be highly local and will only move on to search globally after searching nearlyexhaustively around the current best point. On the other hand, if it is set too high, the search will be excessively global, and the algorithm will be slow to fine-tune any promising solutions."

Expected Improvement

Instead of using a discrete utility function as for the probability of improvement, the work in [63] proposes to use the actual improvement as utility function.

Lemma 3.2. *For the improvement as utility function*

$$u(x_\star) = \max\left(f^{min} - f_\star, 0\right) \tag{3.17}$$

the acquisition can be computed as

$$a^{EI}(x_\star) = \begin{cases} (f^{min} - \mu_\star)\Phi\left(\frac{f^{min}-\mu_\star}{\sigma_\star}\right) + \sigma_\star \phi\left(\frac{f^{min}-\mu_\star}{\sigma_\star}\right) & \text{if } \sigma_\star > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.18}$$

where Φ is the cumulative distribution function and ϕ the density of a standard Gaussian.

Proof. For the proof, we will use a rescaling of the random variable $f_\star$, so that we can rewrite it as

$$f_\star = \mu_\star + \sigma_\star t \tag{3.19}$$

where $t \sim \mathcal{N}(0,1)$ follows a standard Gaussian. Then, we can compute the expectation with respect to t, leading to

$$\begin{aligned} a^{\mathrm{EI}}(x_\star) &= \int_{\mathcal{F}} u(x_\star) p(f_\star \mid \mathcal{H}) \, \mathrm{d}f_\star \\ &= \int_{-\infty}^{\infty} \max\left(f^{\min} - f_\star, 0\right) \phi(t) \, \mathrm{d}t \\ &= \int_{-\infty}^{\frac{f^{\min}-\mu_\star}{\sigma_\star}} (f^{\min} - \mu_\star - \sigma_\star t) \phi(t) \, \mathrm{d}t \\ &= \int_{-\infty}^{\frac{f^{\min}-\mu_\star}{\sigma_\star}} f^{\min} - \mu_\star \, \phi(t) \, \mathrm{d}t \quad - \quad \sigma_\star \int_{-\infty}^{\frac{f^{\min}-\mu_\star}{\sigma_\star}} t \, \phi(t) \, \mathrm{d}t \\ &= (f^{\min} - \mu_\star)\Phi\left(\frac{f^{\min}-\mu_\star}{\sigma_\star}\right) \quad + \quad \sigma_\star \int_{-\infty}^{\frac{f^{\min}-\mu_\star}{\sigma_\star}} -t \, \phi(t) \, \mathrm{d}t \end{aligned} \tag{3.20}$$

where the term on the left already is correct. Now let us inspect the integral on the right

a bit closer

$$\begin{aligned}\int_{-\infty}^{\frac{f^{\min}-\mu_\star}{\sigma_\star}} -t\,\phi(t)\,\mathrm{d}t &= \int_{-\infty}^{\frac{f^{\min}-\mu_\star}{\sigma_\star}} \frac{1}{\sqrt{2\pi}}(-t)\exp\left(-\frac{1}{2}t^2\right)\mathrm{d}t \\ &= \left[\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}t^2\right)\right]_{-\infty}^{\frac{f^{\min}-\mu_\star}{\sigma_\star}} \\ &= \left(\phi\left(\frac{f^{\min}-\mu_\star}{\sigma_\star}\right) - \lim_{s\to-\infty}\exp\left(-\frac{1}{2}s^2\right)\right) \\ &= \phi\left(\frac{f^{\min}-\mu_\star}{\sigma_\star}\right)\end{aligned} \tag{3.21}$$

where we have used first the fact that

$$\frac{\mathrm{d}}{\mathrm{d}t}\exp\left(-\frac{1}{2}t^2\right) = -t\exp\left(-\frac{1}{2}t^2\right) \tag{3.22}$$

and the fact that

$$\lim_{s\to-\infty}\exp(s) = 0\ . \tag{3.23}$$

Finally, substituting the term that we end up with in Equation 3.21 into Equation 3.20 completes the proof. ■

The expected improvement basically consists of two terms, the first one grows whenever the predicted mean $\mu_\star$ decreases, whereas the second term increases for higher uncertainty. In this sense, expected improvement naturally considers both exploration and exploitation. The behaviour of EI is also shown in Figure 3.2, where EI is plotted for both values of improvement $f^{\min}-\mu_\star$ and uncertainty $\sigma_\star$. We can see that the expected improvement not only increases for configurations that offer a large improvement, but also for configurations that have a large uncertainty but maybe offer even a negative improvement.

Entropy based utility

A third alternative is using entropy search to minimize the uncertainty in the area around the global minimizer $x^\star$ as proposed by [57]. The utility then measures the change in entropy of the conditional $p(x^\star\,|\,\mathcal{H})$ and the conditional of choosing the next point as $x_\star$. Formally, it is defined as

$$u(x_\star) = H\left(p(x^\star\,|\,\mathcal{H})\right) - H\left(p(x^\star\,|\,\mathcal{H}, x_\star, f_\star)\right) \tag{3.24}$$

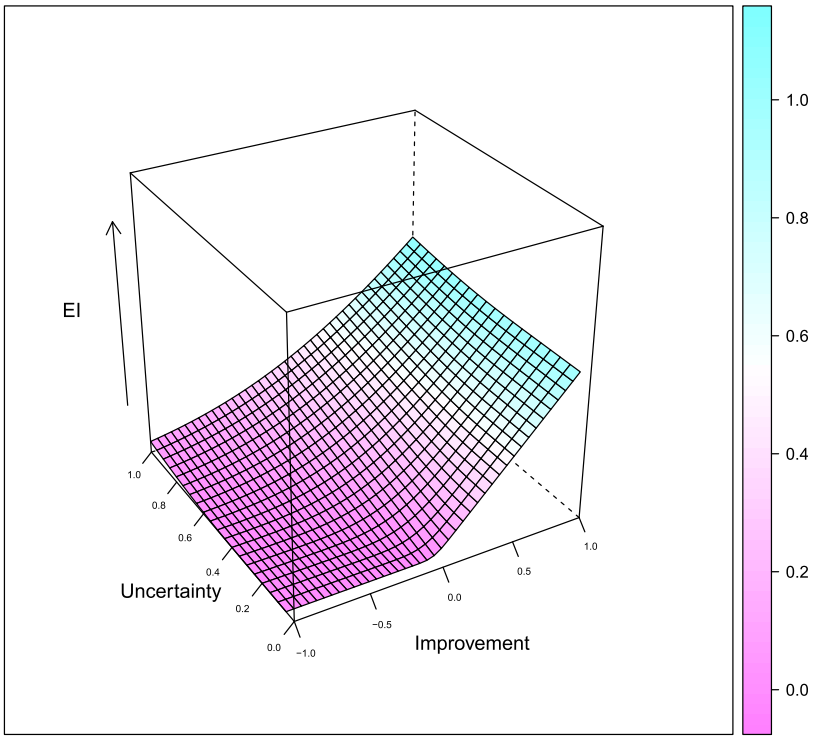

Figure 3.2: Expected Improvement is shown for two inputs, the first axis is the improvement $f^{\min} - \mu_\star$, the axis that is labeled by uncertainty is $\sigma_\star$. As we see, not only configurations with high improvement are considered, as the EI value also increases for uncertain configurations.

where $H(p)$ is the Shannon entropy [83]

$$H(p) = \int p(x) \log(p(x)) \, \mathrm{d}x \quad . \tag{3.25}$$

However, using this utility inside the acquisition function leads to no analytical solution, therefore the integral has to be approximated by sampling methods such as MCMC which is proposed in [33] for example.

Throughout this thesis, we will exlusively employ the expected improvement, as it is analytically computable and seems to have a much better acquisition value than probability of improvement.

3.3 Related Work

We will now discuss the related work in hyperparameter optimization which we firstly divide into two groups. At first, we discuss hyperparameter optimization methods that do not involve Bayesian optimization, those are usually conducting exhaustive search or use genetic algorithms. Afterwards, we will discuss in more detail the related work that is based on sequential model-based optimization, as this will consist of the main competitor methods used in this thesis.

3.3.1 Other Hyperparameter Optimization Methods

There is a variety of different methods that are capable of performing hyperparameter optimization, in this section we will shortly introduce those methods that have a certain downside in comparison to sequential model-based optimization. These downsides are manifold, some methods for instance are computationally inefficient, others are tailored to specific model classes and therefore are not generally applicable. We will start this section by introducing the most commonly employed exhaustive search methods which are grid-search and random search.

Exhaustive Search Methods

From all the methods that have been used, grid-search and its adaptations are probably the most commonly used methods to optimize hyperparameters due to their simplicity. The hyperparameter configuration space Λ is usually the cartesian product of one dimensional configuration spaces Λ_i

$$\Lambda = \prod_i \Lambda_i \quad . \tag{3.26}$$

For all of these subspaces, a discrete subset $G_i \subset \Lambda_i$ is chosen, and the hyperparameter response function F is queried for all configurations λ in the product of these discrete subsets, denoted by G

$$G = \prod_i G_i \tag{3.27}$$

which motivates the name grid-search, although unlike for a real grid, hyperparameter configurations certainly do not need to be chosen equidistantly. A very obvious advantage of grid-search is the fact that besides choosing the subsets G_i the overall procedure is quite simple. However, grid-search is usually computationally inefficient especially

when the dimensionality of Λ is high, as the amount of configurations to evaluate increases exponentially. Unfortunately, since the hyperparameter performance depends on the interaction of hyperparameters, an approach of optimizing each configuration individually does not lead to good results.

Since the user usually wants to make sure to find proper hyperparameters, the grids are chosen to be quite large, which means that in the end a lot of useless computations have been produced. Additionally, grid-search ignores the responses of other hyperparameter configurations and is therefore not adaptive, yet this makes grid-search easily parallelizable. Out of these reasons, it is is usually employed in parallel computing infrastructures such as computer clusters.

There are several extensions of grid-search, a very common one is to adapt to the observed hyperparameter performance by evaluating on a more fine-grained grid in the most promising regions after the performance on the coarse grid has been evaluated. This procedure has for example been applied in the work of [34] and [53].

Another extension of regular grid-search is random search which was introduced by Bergstra and Bengio [7]. Instead of choosing the points to evaluate beforehand, they sample them from a distribution

$$G_i \sim \mathcal{P}_{\Lambda_i} \quad . \tag{3.28}$$

As the authors report, this approach helps in cases of low effective dimensionality, where the hyperparameter performance differs greatly in some dimensions of Λ and is more or less constant in others. However, their method has the downside that one has to define a distribution for every hyperparameter which requires insight in how the final validation performance is influenced by the hyperparameter choice.

Model Specific Methods

There are some methods that work on optimizing hyperparameter performance for a chosen model class, using to some degree the specialities of the model class. Many of these methods are using genetic algorithms, for example the work by de Souza et al. [16] optimizes hyperparameters of a multiclass SVM. Friedrichs and Igel go into a very similar direction in [29]. The work presented in [45] estimates hyperparameters by using gradient descent based optimization on the validation loss with respect to the hyperparameters. This possibility was also discussed earlier but neglected since not every loss is smooth with respect to its hyperparameters, especially when hyperparameters are integers or categorical. A similar approach is used for kernel methods in [82].

Maclaurin et al. also employ hyperparameter gradients to optimize parameters such as learning rates or initializations, expressing the model learned in the end as a sum of SGD updates and then differentiating the validation loss with respect to these hyperparameters. While the approach is very interesting, it is limited to continuous hyperparameters and also those that are represented in the finally learned model.

When the problem at hand is time-series analysis, the work of [61] can be applied, but is not only limited to this scenario. For regression problems when only small amounts of training data are given, the paper by Chapelle et al. [11] can be employed, for log-linear models the efforts by [25] can be implemented. Moreover, the contribution by [59] optimizes hyperparameters in Bayesian topical trend analysis, [44] optimizes hyperparameter configurations for graph-based semi-supervised learning methods, finally for probabilistic prototype-based models the work by [81] may be used.

As we see, there are many approaches of conducting hyperparameter optimization for a specific scenario, may it be model-choice, scenarios where only limited amounts of data are given or assumptions such as continuous hyperparameters. While some of these methods are quite powerful for their respective domain, they cannot be used in general for any kind of hyperparameter configuration as SMBO can.

Learning Curve Prediction

When learning a machine learning model on some data in a sequential manner, the human expert usually can estimate very quickly whether the learning procedure will converge or not, for example by simply observing if the training loss drops as well as the variance in reduction, i.e. by observing the learning curve. If for example the training loss increases and decreases non-systematically, one usually interrupts the learning process and restarts it using for example smaller learning rates. In this way, the hyperparameter space Λ and the response f are explored much faster, as the black-box assumption on running f is ignored. This makes a lot of sense, especially for learning machine learning models in a sequential fashion such as with stochastic gradient descent.

For a couple of years, exploiting learning curves in an automatic fashion in order to speed up hyperparameter optimization in total has gained much attraction. Domhan et al. [18] assume the learning curve to be shaped by a weighted sum of eleven differently parametrized base functions that all have a more or less similar shape. Given some iterations on a training run, they learn the the weights and parameters using MCMC sampling, where they adapt the prior to only consider models that actually decrease the loss. Swersky et al. [88] also include learning curves into their Bayesian optimization

scheme, however, in contrast to [18] they do not stop the learning process but freeze it in favor of starting another, yet going back to the frozen process eventually. As a follow up of their work, [48] merge the ideas by Domhan et al. into a Bayesian neural network that then predicts weights, parameters as well as the final performance of the hyperparameter configuration. As both approaches are learned with MCMC sampling, [2] speeds up this process by learning the learning curve prediction models as simple sequential regression models.

Overall, the inclusion of learning curves in the process of SMBO sound very promising, however, as our own work presented in this thesis focuses solely on predicting response function values, we do not compare our approaches to those using learning curves.

3.3.2 Hyperparameter Optimization based on SMBO

Finally, we will present the major related work that is used in the experiments of this thesis. We will introduce a variety of surrogate models that are used in SMBO-based hyperparameter optimization. The introduction sequence will be given chronologically.

Sequential Model-Based Algorithm Configuration (Independent-RF)

The first approach of using Bayesian optimization for hyperparameter optimization is conducted by Frank Hutter et al. in the seminal paper [39] which introduces SMAC. Hutter proposes to learn random forests on the observed hyperparameter performance in order to predict hyperparameter performance of unseen configurations. The authors argue that learning random forests makes sense when hyperparameters are hierarchical, i.e. setting one hyperparameter to be active introduces a lot of new hyperparameters that need to be tuned.

A random forest is a collection of n many functions $m_1, ..., m_n$ which are all individually learned decision trees or regression trees as for the scenario of hyperparameter optimization. Using the ensemble, uncertainties can be computed by assuming that the collection of functions and their respective predictions for a given configuration stem from a Gaussian distribution, which is the natural distribution we want our surrogate to follow in order to compute expected improvement analytically. The mean and variance of

the collection are computed

$$\mu(f_\star) = \frac{1}{n}\sum_{i=1}^{n} m_i(x) \tag{3.29}$$

$$\sigma^2(f_\star) = \frac{1}{n-1}\sum_{i=1}^{n}(m_i(x) - \mu(f_\star))^2 \tag{3.30}$$

using the formula for empirical mean and variance of a Gaussian. Note that since SMAC is basically a random forest learned only on the target, we also call it independent random forest (I-RF)

Spearmint (Independent-GP)

Another approach of using Bayesian optimization for hyperparameter learning is conducted by Jasper Snoek et al. and is presented in [86]. The authors propose to use the most commonly used surrogate model in Bayesian optimization, which is a Gaussian process. However, the GP is only learned on observations on the target response surface only, in contrast to some of the surrogates using Gaussian processes that were proposed later. Therefore, Spearmint (or I-GP) usually has a weak start, as it does not use meta knowledge, despite Gaussian processes actually being very decent surrogate models.

We will introduce GPs in chapter 6 in more detail, for reference we only give the mean and variance of the surrogate:

$$\mu(f_\star) = k_\star^\top (K + \sigma_y^2 I)^{-1} y \tag{3.31}$$

$$\sigma^2(f_\star) = k_{\star\star} - k_\star \top (K + \sigma_y^2 I)^{-1} k_\star \tag{3.32}$$

which are derived from analysing conditionals of multivariate Gaussians.

Surrogate Collaborative Tuning (SCoT)

The first work to go into the direction of combining meta learning with SMBO-based hyperparameter optimization is the work on *Collaborative Hyperparameter Tuning* (SCoT) proposed by Remi Bardenet et al. in [3]. As they are the first to learn hyperparameter performance across different data sets, they introduce the need of meta features, which are features that describe the data set characteristics. Only by adding these the surrogate model is able to distinguish between observations on different data sets. While doing so, they aslo encounter the problem of differently scaled validation performances. There may be data sets, where the validation accuracy is around 80% and others, where 65%

is already competitive. Simply learning a GP on these observations would make the surrogate think that some configurations are highly performant, although it may only be distracted by different accuracy scales. However, the region in hyperparameter space of well-performing hyperparameters may be similar, hence the motivation of SCoT.

Out of this reason, SCoT aims to learn a ranking SVM on the observed performances, where the rankings follow the performances on each individual data sets only. On the output of this ranking, a Gaussian process is learned to allow for uncertainty estimation. Since SCoT learns a Gaussian process on all of the observations, it becomes inefficient quite early regarding the amount of meta information available. Due to this reason, we cannot present the results of SCoT on all of our experiments conducted.

Multiple Kernel Learning (MKLGP)

The work by Yogatama and Mann [96] as well as SCoT identifies the problem of differently scaled validation performances, but instead of learning a slow ranker, they decide to scale the observed performances for each data set and the ones for the target problem on-the-fly. They then introduce a kernel that is a convex combination of two individual kernels as

$$k(x_i, x_j) = \alpha k_1(x_i, x_j) + \beta k_2(x_i, x_j) \qquad \alpha + \beta = 1 \ . \tag{3.33}$$

The first kernel k_1 is the ARD Gaussian kernel that measures the similarity between hyperparameters on each dimension differently, by learning the individual scales. For more detail on kernels, we refer the reader to chapter 6.

The second kernel that Yogatama and Mann employ is a kernel that measures distances between data sets, and is defined as

$$k_2(x_i, x_j) = \begin{cases} 1 - \|x_i - x_j\|_2^2 & \text{if } x_j \text{ is among the nearest neighbors of } x_i \\ 0 & \text{otherwise} \end{cases} \tag{3.34}$$

where the nearest neighbor relationship is computed by using the Euclidean distance on the meta features. Therefore, if both configuratons x_i and x_j stem from the same data set, the contribution to the similarity is β, if not, it is either zero or a discounted β, where the discount factor depends on the meta features.

Given all the meta data, the approach we call MKLGP learns a full Gaussian process on the data, however uses a more refined kernel function in order to account for differently scaled validation performances.

CHAPTER 4

EXPERIMENTAL SETUP

Before we are able to present our contributions to the problem of hyperparameter optimization across data sets in the upcoming chapters, we need to describe the meta data sets consisting of hyperparameter performance observations on a set of data sets. This chapter covers the creation of the meta data sets, as well as a detailed introduction of the evaluation measures which are used to evaluate the performance of a hyperparameter optimization strategy.

4.1 Meta Data Creation

This section describes the creation of our meta data sets. In total, we have created four meta data sets, where three of these meta data sets consist of hyperparameter performance for (multiclass) classification tasks, and the fourth meta data set contains experiments across classification and regression tasks.

For the first three meta data sets, we considered the classification problem, where the target space consists of $C \geq 2$ many classes

$$\mathcal{Y} = \{1, \ldots, C\} \ . \tag{4.1}$$

As our target metric, we choose accuracy which counts the number of correctly classified instances. For a data set D and a model m, accuracy is defined as

$$\text{ACC}(m, D) = \frac{1}{N} \sum_{(x,y) \in D} \chi(m(x) = y) \tag{4.2}$$

where χ is an indicator function that returns one when its predicate is true and zero otherwise. In this way, all correctly classified instances are counted. As we have defined the hyperparameter optimization problem as finding the correct hyperparameters to minimize a risk, we can maximize accuracy by minimizing misclassification rate

$$\mathrm{MR}(m,D) = 1 - \mathrm{ACC}(m,D) \ . \tag{4.3}$$

For regression problems, the target space becomes the line of real numbers $\mathbb{R}$, and we use the root mean squared error (RMSE) as loss function in our optimization problem:

$$\mathrm{RMSE}(m,D) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(m(x_i) - y_i)^2} \tag{4.4}$$

In the following, we will describe in detail how we have computed the four meta data sets that are being used in this thesis, by describing the data sets used, the models and respective hyperparameters that have been used. Finally, we discuss also the meta features that we have computed for each data set.

4.1.1 AdaBoost Meta Data

The first, and arguably most simple meta data set is constructed by using the famous AdaBoost algorithm, which was invented by Freund and Schapire in [26]. AdaBoost learns a weighted sum of K weak learners, precisely the prediction is

$$m^K(x) = \sum_{i=1}^{K} m_i(x) \tag{4.5}$$

where all m_i are simple machine learning models that can be learned without much effort. For AdaBoost, decision trees have been employed as weak learners. Boosting works by fixing all previously learned weak learners into a model m^{t-1}, and then optimizing the next weak learner to directly reduce the error that the ensemble produces

$$\ell(m^t) = \ell(m^{t-1} + \alpha_t m_t) \ . \tag{4.6}$$

In this way a set of diverse models is found, as each new weak learner directly minimizes the error the old learners have done, and therefore considers different aspects of the problem. We use the implementation from `http://www.multiboost.org`, where we consider two hyperparameters, at first the number of iterations I, secondly, the number of weak learners M. The number of iterations was evaluated for

$$I \in \{2,5,10,20,50,100,200,500,1000,2000,5000,10000\} \tag{4.7}$$

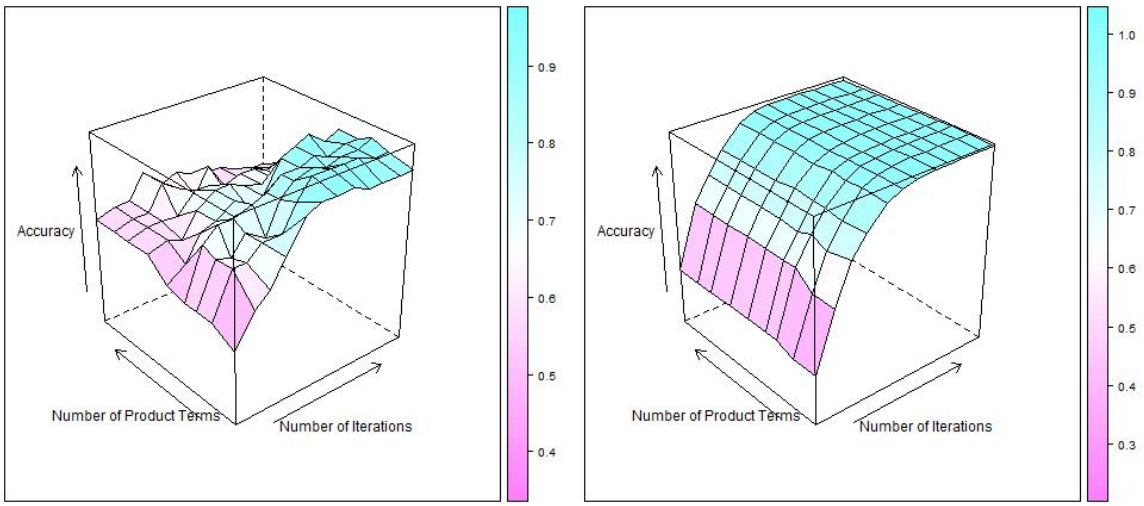

Figure 4.1: AdaBoost response surfaces for automobile (left) and pendigits (right). For the automobile data set, many iterations but a low number of weak learners seems to behave best, for pendigits, the number of weak learners has no influence.

and the number of base models was analyzed for

$$M \in \{2,3,4,5,7,10,15,20,30\} \quad . \tag{4.8}$$

We have merged existing splits into one and then taken 80% of the data as training data, and the remaining 20% as test data. The meta data set that we create this way is rather small, since $\dim(\Lambda) = 2$, overall we obtain 108 meta instances per data set that we try to solve. Four cherry-picked response surfaces of the AdaBoost meta data can be seen in Figures 4.1 and 4.2, that all show a different behaviour regarding hyperparameter choices.

4.1.2 SVM Meta Data

Following the work of Bardenet et al.[3], we have created a meta data set using the famous support vector machine classifier which was introduced in 1995 by Cortes and Vapnik [15]. A support vector machine finds a hyperplane as decision boundary in the vector space of the data. The final decision has the form

$$m(x_i) = \text{sign}\,(\theta_0 + \theta^\top x_i) \qquad \theta_0 \in \mathbb{R} \quad \theta \in \mathbb{R}^d \quad . \tag{4.9}$$

The optimal decision boundary (θ_0, θ) is then found by maximizing the margin of the hyperplane to all the support vectors. This is accomplished by solving the following

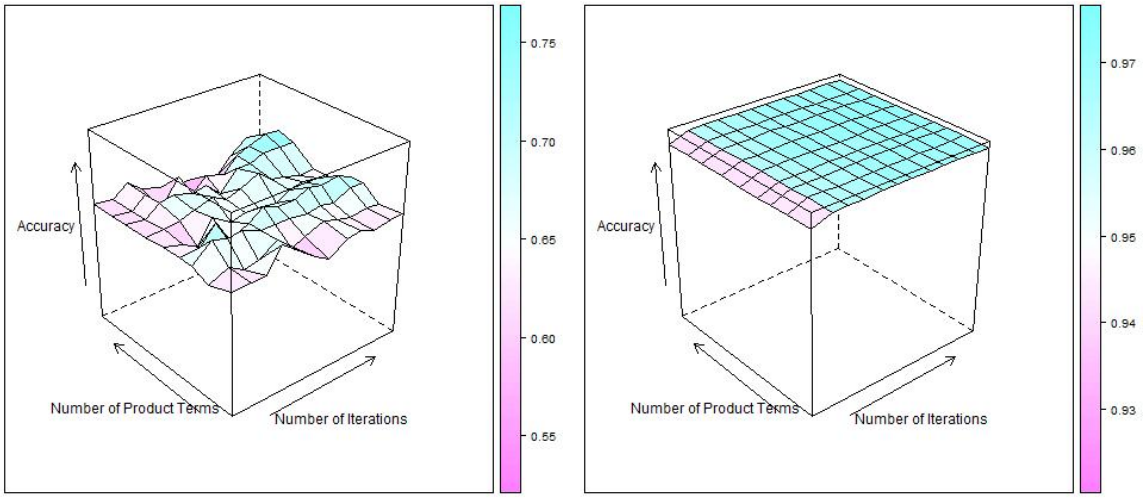

Figure 4.2: AdaBoost response surfaces for bupa (left) and svmguide1 (right). For bupa, the hyperparameter performance seems to differ more or less randomly as no clear pattern is visible, for svmguide1, the choice of hyperparameters has next to no influence.

minimization problem

$$\begin{aligned} \min_{\theta} \quad & \frac{1}{2}\|\theta\|_2^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i(\theta_0 + \theta^\top x_i) \geq 1 - \xi_i \end{aligned} \tag{4.10}$$

where ξ_i are the so called slack variables, which allow a misclassification in the training data in cases where it is not linearly separable. The associated parameter $C > 0$ is a hyperparameter that specifies how much a misclassified data instance costs.

SVMs can also be solved in the dual problem [38], using Lagrange multipliers and the Karush-Kuhn-Tucker conditions [51] to compute the hyperplane parameters, leading to the dual SVM formulation. Using the duality approach, kernels are employed to learn nonlinear decision boundaries using the kernel trick. The final prediction is then computed as

$$m(x_i) = \text{sign}\left(\sum_j \theta_j y_j k(x_i, x_j) + \theta_0\right) \quad . \tag{4.11}$$

Our meta data consists of the validation performance of differently hyperparametrized versions of support vector machines. For 120 binary or multiclass classification data sets, we have merged the existing splits into one data set, afterwards we have split the data into a training part which contains 80% of the training instances and a validation

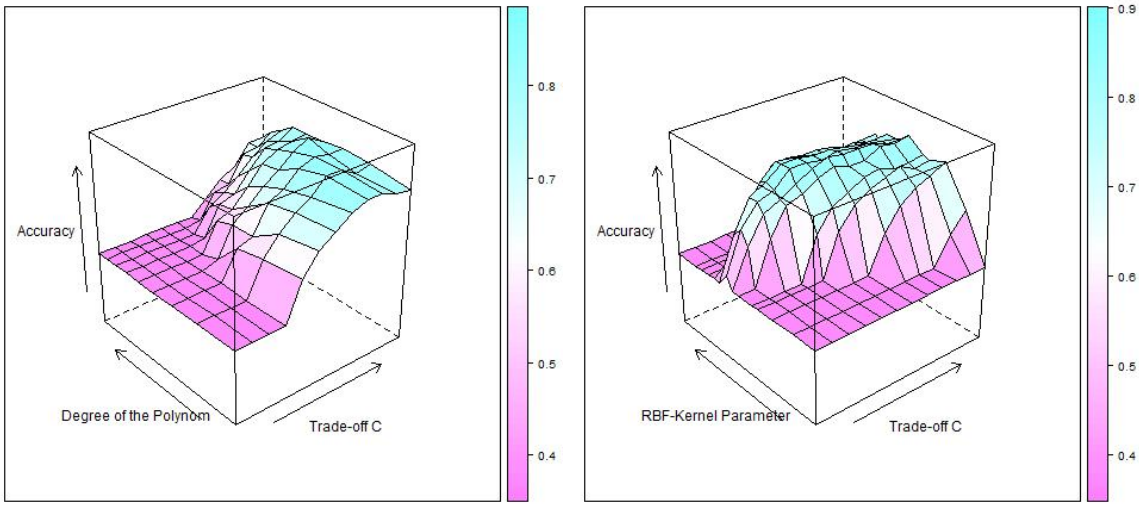

Figure 4.3: SVM response surfaces for the polynomial (left) kernel and the Gaussian (right) kernel for the ecoli data set.

partition which holds the remaining 20% of the data. We have then trained the SVM on the training data and have made predictions for the validation data, where we measure the misclassification rate.

The kernel choice that has to be dealt with when learning SVMs is the first hyperparameter, concretely we have chosen to assess the performance of three different kernels. At first, we employed the linear kernel

$$k(x_i, x_j) = x_i^\top x_j \tag{4.12}$$

which usually does not yield powerful models. Secondly, we used the polynomial kernel

$$k(x_i, x_j) = (b + x_i^\top x_j)^d \tag{4.13}$$

which accomplishes a nonlinearity through the exponentiation of the linear kernel. Lastly, we have used the Gaussian kernel which is defined as

$$k(x_i, x_j) = \exp\left(\frac{\|x_i - x_j\|_2^2}{2\gamma^2}\right) . \tag{4.14}$$

The choice of kernel thus introduces three binary parameters, which we describe using a one-hot encoding. In this regard, we are already designing a small experiment that resembles a model choice, since employing different kernels also results in different models that are learned. The next two hyperparameters are hierarchical kernel

hyperparameters, which means that they only exist for a specific kernel choice. For the polynomial kernel we consider the degree d of the kernel, which has been evaluated for:

$$d \in \{1,2,3,4,5,6\} \ . \tag{4.15}$$

For the Gaussian kernel we regard the kernel width γ as hyperparameter. It has been set to

$$\gamma \in \{0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000\} \ . \tag{4.16}$$

Whenever a kernel is not active, the respective hyperparameter is set to zero in the meta data set. The final hyperparameter is the cost of the slack variables, which is denoted by C as in the optimization problem stated above. It is our final hyperparameter and has to be specified independent of the kernel. We have used values of C in the range

$$C \in \{2^{-5}, ..., 2^{6}\} \ . \tag{4.17}$$

Finally, for each data set, we arrive at a grid size of 288 hyperparameter configurations that need to be evaluated. Figure 4.3 shows the response surface of using a polynomial and a Gaussian kernel on the ecoli data set.

4.1.3 Weka Meta Data

As we do not only want to learn hyperparameter performance across data sets but also across other problem instances such as model choice, we have created a third meta data set. Similar to the previously introduced meta data sets, it considers classification problems and is called the Weka meta data. It was created using Weka [31], which is a tool that offers a range of different machine learning models. For 59 data sets, we have used Weka to obtain the hyperparameter performance of a total of 19 different classifiers such as Bayesian networks, naïve Bayes, a logistic regression, a support vector machine, neural networks, and even simple classifiers such as a decision stump. An overview over all different classifiers and their respective hyperparameter settings can be seen in the following Table 4.1.

Table 4.1: The grid used to create the Weka meta-data set (Part 1).

Bayes-Net	
estimator	SimpleEstimator
searchAlgorithm	HillClimber, K2, LAGDHillClimber, TabuSearch, TAN
scoreType	BAYES, BDeu, MDL, AIC
Naive Bayes	
No hyperparameters	
Logistic	
maxIts	10, 50, 100, 200, 500, 1000
ridge	0, 1E-8, 1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 0.1, 0.5, 1, 5, 10
MultiLayerPerceptron	
hiddenLayers	1
learningRate	0.001, 0.01, 0.1, 0.25, 0.5
momentum	0.001, 0.01, 0.1, 0.25, 0.5
trainingTime	10, 50, 100, 250, 500, 1000
validationSetSize	0, 2, 4, 6, 8, 10
validationThreshold	5, 10, 20, 50, 100
SMO	
c	0.03125, 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32
kernel	PolyKernel, RBFKernel, Puk
gamma	0.001, 0.01, 0.1, 0.25, 0.5
omega	0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000
sigma	0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000
degree	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
useLowerOrder	True, False
SimpleLogistic	
weightTrimBeta	0, 0.0001, 0.001, 0.01
heuristicStop	0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
maxBoostingIterations	100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
useAIC	True, False
useCrossValidation	True, False

Table 4.2: The grid used to create the Weka meta-data set (Part 2).

IBk	
KNN	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50
distanceWeighting	No distance weighting, Weight by 1/distance, Weight by 1-distance
KStar	
globalBlend	0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100
entropicAutoBlend	True, False
DecisionTable	
search	BestFirst, GreedyStepwise
conservativeForward	True, False
direction	Forward, Backward
searchTermination	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
JRip	
folds	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
minNo	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
optimizations	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
usePruning	True, False
OneR	
minBucketSize	1, 2, ..., 50
PART	
confidenceFactor	0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4
minNumObj	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
numFolds	2, 3, 4, 5, 6, 7, 8, 9, 10
Pruning	ReducedErrorPruning, Pruning, Unpruned
ZeroR	
No hyperparameters	
DecisionStump	
No hyperparameters	

Table 4.3: The grid used to create the Weka meta-data set (Part 3).

J48	
confidenceFactor	0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4
minNumObj	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
numFolds	2, 3, 4, 5, 6, 7, 8, 9, 10
Pruning	ReducedErrorPruning, Pruning, Unpruned
subtreeRaising	True, False
useLaplace	True, False
LMT	
convertNominal	True, False
minNumInstances	2, 4, 6, 8, 10, 12, 15, 17, 20, 25, 50
splitOnResiduals	True, False
useAIC	True, False
weightTrimBeta	0, 0.0001, 0.001, 0.01
RepTree	
minNum	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
minVarianceProb	0, 1E-8, 1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 0.1, 0.5, 1
numFolds	2, 3, 4, 5, 6, 7, 8, 9, 10
noPruning	True, False
RandomForest	
maxDepth	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
numFeatures	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
numTrees	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 500
RandomTree	
maxDepth	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
minNum	0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, 9, 10
numFold	0, 2, 3, 4, 5

Following this huge table of models and associated hyperparameter configurations, we end up having 21,871 configurations to evaluate for each data set. For a number of 59 individual data sets, this sums up to nearly 1.3 million experiments which have been conducted.

4.1.4 Multilayer Perceptron Meta Data

The final contribution of this thesis is to learn hyperparameter performance in a surrogate model across different problem tasks such as regression and (multiclass) classification. To accomplish this goal, the experiments need to be conducted using a prediction

model that is capable of solving both tasks while using the same or nearly the same hyperparameter space. As a negative example, we could not try to learn across tasks when we use an SVM for classification and a factorization machine for regression as both models behave totally different and do not share a single hyperparameter besides learning rates and regularization parameters.

Out of this reason, we choose multilayer perceptrons as prediction model, for a detailed introduction of how they work, we would like to refer the reader to chapter 5 where MLPs are introduced. MLPs are able to solve any task the only task-dependent part of the model is the activation function used within the last layer. Every other hyperparameter is task-independent and therefore generalization can be made. Additionally, since MLPs have a relatively large hyperparameter space, making use of the hyperparameter performance on other tasks is a promising direction.

We will now discuss which parameters of a neural network we have treated as hyperparameter, in general we can cluster the hyperparameters into three different groups:

Structure-based hyperparameters In principal, the structure of a neural network is given by any directed acyclic graph (DAG); however, usually assumptions are made that simplify the structure, as the space of all DAGs is too large to search through in an efficient manner. Therefore, feedforward neural networks assume a layer structure, where the l-th layer neurons only depend on the neurons in layer $l-1$, disallowing any form of skip connections. On top of this, we make two more assumptions that simplify the structure. The number of layers will be denoted by L and the number of neurons will be denoted by N_l, we make a simplificiation by assuming that N_l amounts to the same value for all layers, to avoid the hyperparameters being set hierarchically. Additionally, the activation function being used is a hyperparameter, where one can choose between a sigmoid function, tanh or activation functions based on rectified linear units such as the standard ReLU or leaky ReLU. We again slightly simplify this choice by assuming that $\sigma_l \equiv \sigma$ is independent of the layer, except for the output layer, where the activation function is defined by the task to be solved.

Optimization-based hyperparameters As MLPs are highly non-convex, the choice of optimization algorithms and their associated hyperparameters is crucial. Out of this reason they form the second group of hyperparameters. For the optimization algorithm there is a plethora of choices. At first, one may chooses to use plain SGD [9] along

with a constant step-siz or with some step size adaptation techniques such as the bold-drivers rule [4] where the step size is halved if the loss on the training data increases, or increased by a small factor if the loss decreases. Another technique that simply shrinks learning rates is annealing which was presented in [73]. More sophisticated techniques however try to adapt the learning rate for each parameter specifically where momentum based methods [67] are used such as AdaGrad [21], AdaDelta [97], ADAM [47] and its Nesterov extension NADAM [19]. For all the variants that were tested in our experiments, for the initial learning rate we have used the suggestions by the respective publications.

Additionally, the number of epochs that are conducted until learning is finished is a hyperparameter that has a high influence on the final performance. Usually, one does not want to learn too short, but also does not want to overfit by learning too much. Finally, the number of data instances that is used for one update of the model parameters - the batch size - is also a hyperparameter that needs to be specified.

Regularization-based hyperparameters As neural networks are powerful prediction models, one needs to take good care to prevent them from overfitting to the signal in the training data. Because of this, the choice of regularization is crucial to the generalization abilities of a model. L_p regularization is a commonly used strategy, for $p = 2$ this is usually called Tikhonov regularization [90] or Ridge Regression [37]. When dealing with sparse data, LASSO regularization [89], where $p = 1$ is used to obtain sparse solutions that conduct feature selection automatically. Specifically for neural networks, techniques such as dropout [87] are used where the signal of a neuron is randomly dropped out with a given probability, this technique is sometimes humourously called optimal brain damage. In our experiments, we restrict ourselves to these three methods of regularization although more techniques exist, such as early stopping for instance.

Table 4.4: Overview of the hyperparameter grid used to create the MLP meta data set

Structure	**Values:**		
Activation Function	ReLU	leakyReLU	tanh
Number of Layers	5	10	20
Number of Neurons	10	20	50
Optimization			
Optimizer	Adam	AdaGrad	AdaDelta
Number of Epochs	10	100	
Regularization			
Dropout	0	0.2	0.4
L_p Regularization	L_1	L_2	
Regularization Constant	0.01	0.001	0.0001

The hyperparameter grid that shows the different settings per hyperparameter can be seen in Table 4.4. Overall, for one data set we have a total of 2916 different individual experiments. We use a mixture of binary classification, multiclass classification and regression data sets amounting to 70 data sets that have been downloaded mainly from the UCI repository. The distribution of data sets is as follows. We conduct experiments on 44 binary classification problems, 12 multiclass classification problems and 14 data sets that comprise a regression task. In total, the meta data consists of 204,120 experiments. To conduct these experiments, we use Keras [13] which is a wrapper that builds on Tensorflow/Theano and allows the user to build neural networks with more simplicity than using Tensorflow/Theano as standalone.

4.2 Meta Features

In order to generalize hyperparameter performance across problem instances such as data sets, we need to somehow inform the surrogate model of the data set it is currently working on. Otherwise, we have some meta instances with same features, but different target value, which would make learning of the surrogate next to impossible.

Table 4.5: The list of all meta-features used in our classification tasks.

Meta-Features	
Number of Classes	Class Probability Max
Number of Instances	Class Probability Mean
Log Number of Instances	Class Probability Standard Deviation
Number of Features	Kurtosis Min
Log Number of Features	Kurtosis Max
Data Set Dimensionality	Kurtosis Mean
Log Data Set Dimensionality	Kurtosis Standard Deviation
Inverse Data Set Dimensionality	Skewness Min
Log Inverse Data Set Dimensionality	Skewness Max
Class Cross Entropy	Skewness Mean
Class Probability Min	Skewness Standard Deviation

Out of this reason, the so called meta features are being added to the meta data, where the meta features $m(D)$ describe characteristics of a data set. A very simple, but non-informative meta feature could simply be a one-hot encoding that only determines the data set we are considering

$$m(D_i) = \left(\chi(j = i)\right)_{j=1}^{|\mathcal{D}|} \tag{4.18}$$

so, if we for instance have 10 data sets, and we are currently working with the seventh data set, we would characterize this by a vector of zeros, where only the seventh entry is set to one.

4.2.1 Task-dependent meta features

As one may imagine, these features are simple, but are not really expressive, as they only indicate which data set is used and contain no further information. To increase the expressiveness of our meta data sets, we have used a diverse set of meta features, which can be seen in Table 4.5. In the following, we will define all meta features for a data set

$$D = \{(x_1, y_1), ..., (x_N, y_N)\} \qquad x_i \in \mathbb{R}^d \quad y_i \in \{1, ..., C\} \tag{4.19}$$

where the *number of classes* is denoted as C, the *number of instances* is N and its respective logarithm. The *number of features* is simply d, where we also include its logarithm. The *data dimensionality* is computed by dividing the number of features by the amount of data instances, so d/N, where we also include its inverse and the

logarithm of both values. A measure that resembles the disorder of classes is the *class cross entropy* which is computed as

$$\text{entropy}(D) = -\sum_{c=1}^{C} p(c) \ln p(c) \tag{4.20}$$

where $p(c)$ is the relative frequency of class c in the dataset, and is computed as

$$p(c) = \frac{1}{N} \sum_{i=1}^{N} \chi(y_i = c) \ . \tag{4.21}$$

For a uniform distribution of classes, i.e. $p(c) = 1/C$, the entropy attains its maximum value and becomes minimal for the case where only one class is present in all data instances. Entropy therefore can be understood as a measure that characterizes skewed distribution of labels, if its value is low one may consider for example using a pairwise loss between instances, such as the Bayesian Personalized Ranking (BPR) loss [70], to account for skewed label distributions. The *class probability max (min)* are simply the maximum (minimum) values of $p(c)$ over all classes, in addition, we compute their *mean and variance*.

4.2.2 Data-dependent meta features

In the previous subsection, we have only specified meta features that depend on the underlying task to be solved, such as meta features involving the class distribution as we have seen.

Considering the feature values, we compute the *skewness* of a feature j as

$$\text{skewness}(j) = \frac{N}{(N-1)(N-2)} \sum_{i=1}^{N} \left(\frac{x_{ij} - \mu(j)}{\sigma(j)} \right)^3 \tag{4.22}$$

where x_{ij} is the j-th feature of the i-th instance, $\mu(j)$ is the mean and $\sigma(j)$ is the standard deviation of the j-th feature over all data instances. We then take the *maximum, minimum, mean and standard deviation* over the kurtosis values of all features in the data set, to obtain four more meta features, that express how the features values are distributed in the data set, and how much this varies among the features. We repeat the procedure with the *kurtosis*, which is defined as

$$\text{kurtosis}(j) = \frac{N-1}{(N-2)(N-3)} \sum_{i=1}^{N} (N+1) \left(\left(\frac{x_{ij} - \mu(j)}{\sigma(j)} \right)^4 + 6 \right) \tag{4.23}$$

and measures the curvature of the feature value distribution. As for the skewness, we compute the kurtosis values for all features and then include their *maximum, minimum, mean and standard deviation* in the meta data.

Some of these features cannot be computed for regression problems, since they are dependent on a classification problem such as the *number of classes*. Out of this reason, we have omitted all task-specific meta features when creating the MLP data set, the resulting table of meta features that were included can be seen in Table 4.6. Two alternative approaches can be thought of. At first, we could compute them for the classification problems and simply set them to zero for the regression problems, however, this likely introduces a bias that confuses the surrogate model. Secondly, we can think of similar meta features for the regression problems, such as using the size of the interval of y values, or compute a binning of target values to then compute something like the entropy of the targets.

Table 4.6: The list of all meta-features used for the MLP meta data

Meta-Features	
Number of Instances	Kurtosis Min
Log Number of Instances	Kurtosis Max
Number of Features	Kurtosis Mean
Log Number of Features	Kurtosis Standard Deviation
Data Set Dimensionality	Skewness Min
Log Data Set Dimensionality	Skewness Max
Inverse Data Set Dimensionality	Skewness Mean
Log Inverse Data Set Dimensionality	Skewness Standard Deviation

As a final processing step, we have scaled all meta features to a standard Gaussian, since some of the meta features, such as the number of instances vary a lot among the data sets and then give the surrogate model a hard time to learn from them.

4.3 Evaluation Measures

To assess the performance of our hyperparameter optimization strategies, we need to have metrics that measure how well we are doing in each trial. In this section, we will introduce several evaluation measures, where each metric covers a different aspect of the evaluation. In all our experiments, we usually train $R \in \mathbb{N}$ many surrogate models to reduce random effects such as the initalization of the surrogate models. Additionally, especially in the very first trials it may happen that the value of the acquisition function is equal for some hyperparameter configurations and since we have to make a decision ties are broken arbitrarily. As the early trials usually determine the further optimization significantly, we need to cancel out these random effects.

Furthermore, for all of our experiments unless not stated otherwise explicitly, we have scaled the hyperparameter performance values to the intervall $[0,1]$. This can be accomplished by assigning zero to the lowest accuracy - or highest RMSE for regression problems - and one to the highest accuracy. More specifically, let us define

$$f_D^{\max} = \max_{\lambda \in \Lambda} f_D(\lambda) \qquad f_D^{\min} = \min_{\lambda \in \Lambda} f_D(\lambda) \tag{4.24}$$

where we assume that the respective minimum and maximum are unique. Of course, this is not the case in reality, however, the notation is more natural than having a set of equally working configurations and picking one. Having defined these extrema, we can scale accuracies as:

$$\tilde{f}_D(\lambda) = \frac{f_D(\lambda) - f_D^{\min}}{f_D^{\max} - f_D^{\min}} \tag{4.25}$$

and similarly for regression problems:

$$\tilde{f}_D(\lambda) = 1 - \frac{f_D(\lambda) - f_D^{\min}}{f_D^{\max} - f_D^{\min}} \quad . \tag{4.26}$$

In this way, the scaled performances are in the same interval for all problem tasks and for all data sets. However, we will not continue to use $\tilde{f}$ but continue to simply use f to avoid unnecessary clutter.

4.3.1 Average Hyperparameter Rank

In an ideal world, from the discrete set of hyperparameter configurations considered, we would always choose the configuration that simply has the lowest validation error. Conversely, if we pick the worst hyperparameter configuration initially, then this should be also covered in the evaluation metric. Let us denote by

$$\lambda^{\text{best}}(\Psi, D, t) = \arg\min_{\lambda \in \Lambda_t(\Psi)} f_D(\lambda) \tag{4.27}$$

the best hyperparameter configuration that Ψ has found on data set D up to trial t from the set of all considered hyperparameters $\Lambda_t(\Psi)$. Then by

$$\text{rank}_D(\lambda) = \left| \{ \lambda' \in \Lambda \mid f_D(\lambda') \leq f_D(\lambda) \} \right| + 1 \tag{4.28}$$

we assign ranks to hyperparameter configurations based on how well they perform. In this formulation, the best hyperparameter will have rank one.

Now we are able to define the *average hyperparameter rank* at trial t which is simply the rank of the best hyperparameter configuration found by Ψ, averaged over all data sets D and over all random runs R

$$\text{avgHypRank}(\Psi, \mathcal{D}, t) = \frac{1}{R} \frac{1}{|\mathcal{D}|} \sum_{r=1}^{R} \sum_{D \in \mathcal{D}} \text{rank}_D \left(\lambda^{\text{best}}(\Psi_r, D, t) \right) \tag{4.29}$$

where Ψ_r denotes the r-th experiment using the surrogate Ψ. The lower the value of the average hyperparameter rank, the better the performance of Ψ. However, the average hyperparameter rank also has some issues, as it sums over all data sets D with equal weight, it may be the case that simpler hyperparameter optimization problems that are easy to solve by even a random surrogate model occlude the real performance of Ψ. Additionally, the performance improvement is also not considered, in some cases a better rank may mean only a small performance lift in reality, while in other cases the difference may be much bigger.

4.3.2 Average Rank among Competitors

Contrary to the average hyperparameter rank defined above, the *average rank* measures the surrogate's performance a little differently. First of all, the average rank compares different surrogate models directly as competitors. So we assume we have $C \in \mathbb{N}$ many different surrogates competing, then let us define the performance of a specific surrogate Ψ^c at trial t as:

$$\text{perf}\left(\Psi^c, t\right) = \frac{1}{R} \frac{1}{|\mathcal{D}|} \sum_{r=1}^{R} \sum_{D \in \mathcal{D}} f_D \left(\lambda^{\text{best}}(\Psi_r^c, D, t) \right) \ . \tag{4.30}$$

In words, the performance of a surrogate at time t is the performance of the best working hyperparameter configuration so far, averaged over all R many experiments and all data sets. Having defined the performance, we can now write down the *average rank* of surrogate Ψ^c among all competing methods $\Psi^{c'}$ on trial t as

$$\begin{aligned} \text{avgRank}(\Psi^c, t) = 1 &+ \left| \left\{ c' \,\middle|\, \text{perf}\left(\Psi^{c'}, t\right) > \text{perf}\left(\Psi^c, t\right) \right\} \right| \\ &+ \frac{1}{2} \left| \left\{ c' \mid \text{perf}\left(\Psi^{c'}, t\right) = \text{perf}\left(\Psi^c, t\right) \right\} \right| \ . \end{aligned} \tag{4.31}$$

The average Rank consists of two components, at first we have the normal ranking, that simply checks how many models among the competing surrogates show a better performance at trial t. The second component evaluates how many models show the same performance and then assigns the same average rank to all of these models. As a small

example, the reader may consider five surrogates $\Psi^1, ..., \Psi^5$ that have performances associated as $0.8, 0.7, 0.7, 0.7, 0.4$. In this case, the first surrogate would simply get rank one, as there are no better and no equivalent models, the last surrogate Ψ^5 is clearly the worst and therefore would be ranked last, so obtain a rank of five. The surrogates Ψ^2, Ψ^3, Ψ^4 all show the same performance, they would normally occupy the ranks two, three and four, as we assign the average rank they would then all be ranked on position three.

4.3.3 Average Distance to the Minimum

In addition to the previous evaluation metrics, we introduce the simplest evaluation measure, which is the average distance to the minimum. For a surrogate Ψ at trial t, the average distance to the minimum is the difference in performance of the best configuration found so far and the best configuration on the data set, averaged over all random runs and data sets

$$\text{avgDistToMin}(\Psi, \mathcal{D}, t) = \frac{1}{R} \frac{1}{|\mathcal{D}|} \sum_{r=1}^{R} \sum_{D \in \mathcal{D}} \left(f_D \left(\lambda^{\text{best}}(\Psi_r, D, t) \right) - f_D^{\min} \right) \quad . \tag{4.32}$$

We could also use the absolute value of the difference, but since the minimum is always smaller (or equal) than the first expression, the distance is always positive. Naturally, small values imply good performance of the surrogate Ψ.

4.3.4 Fraction of Unsolved Data Sets

Finally, the last evaluation metric that we want to suggest is the fraction of unsolved data sets. At trial t, it measures over all data sets being the target once the amount of data sets, where the minimum has been found already. Formally, we define the fraction of unsolved data sets of a surrogate Ψ at trial t over R many random runs as:

$$\text{fracUnsolvedDataSets}(\Psi, \mathcal{D}, t) = 1 - \frac{1}{R} \frac{1}{|\mathcal{D}|} \sum_{r=1}^{R} \sum_{D \in \mathcal{D}} \chi \left(f_D \left(\lambda^{\text{best}}(\Psi_r, D, t) \right) = f_D^{\min} \right) \quad . \tag{4.33}$$

This means that we sum up over those data sets and random runs where we have already found the minimum, while of course averaging the result to make it independent of the size of $\mathcal{D}$ and the amount of random repetitions.

CHAPTER 5

Relational Hyperparameter Optimization with Factorized Multilayer Perceptrons

Relational Learning has proven to be a very powerful machine learning tool for areas where a set of entities and observed relations between these entities are given, for example in recommender systems in e-commerce. The release of the Netflix prize challenge in 2007 [6] has greatly influenced the research on machine learning models for recommender systems. In recommender systems, the target relation is usually only partially observed between entities, and the goal is to predict whether a not yet observed relation between entities exists. For e-commerce applications such as a web shop, two sets of entities are for instance users that are customers of the shop and the set of items that can be bought in the respective shop. In many applications, the target relation is binary and simply encodes whether a user has bought a certain item or not. However, the target relation may also be continuous, for example when modeling the user's interest in an item through a rating system, such as in the Netflix prize, where a five-star rating system was used. By correctly predicting the target relation for unobserved entity pairs, stakeholders of such e-commerce are enabled to deliver both precise and user-adaptive recommendations for each customer individually.

In this chapter, we will present a surrogate model that is inspired by a famous model in recommender systems, the factorization machine that was invented by Rendle [69]. By designing such a surrogate, we are able to learn latent embeddings from categorical

variables, such as the data set indicator variable. In this way, we aim to learn data set representations automatically. In order to do so, we first cover a small overview of relational models being applied on matrix completion problems, to give an understanding of how the intuition can be carried over to surrogate models for Bayesian hyperparameter optimization.

5.1 Relational Models in Recommender Systems

Relational data is usually comprised of a set of R many partially observed relations on a set of entities [20], which will be denoted by $\mathcal{E}$. The data for relation r denoted by D_r consists of observed tuples $e_r \in \mathcal{E}^{n_r}$ of arity n_r and an observed value $y_r \in \mathbb{R}$, which is the value that we want to predict. For unobserved tuples, if the relation r is called the target relation, as mentioned in the introduction, in many scenarios y_r is binary and simply decodes if the relation between the entities exists, or not. Additionally, for many relations the arity is $n_r = 2$, as in the web shop example, where only a relation between a user and an item is observed, and not a relation between a user, an item and another item for instance. Finally, as the set $\mathcal{E}$ models all possible entities, we denote by $E_r \subset \mathcal{E}^{n_r}$ the set of actually used entities for relation r. For example, the tree in my garden might have a relation with some birds building their hive on it, but certainly these entities are not relevant for the web shop example that we have given above.

The ultimate goal of relational learning is then to find R many models $m_r : E_r \longrightarrow \mathbb{R}$ for each $r \in 1,...,R$ such that the loss function $\ell_r : P(E_r \times \mathbb{R}) \times \mathbb{R} \longrightarrow \mathbb{R}_+$ is being minimized on some test data that is distinct to the training data. In other words, we will seek to minimize

$$\text{error}((D_r^{\text{test}})_{r=1}^R) = \sum_{r=1}^R \alpha_r \ell_r(D_r^{\text{test}}, m_r) + \text{Reg}(m_r) \ . \tag{5.1}$$

The term $\text{Reg}(m_r)$ is a regularization term that is used to prevent the model from overfitting to the training data. By $P(A)$ we denote the power set of A, namely the set of all subsets of A, as each possible relational data set is then an element of $P(E_r \times \mathbb{R})$. The coefficients $\alpha_r \in [0,1]$ define, how strong the influence of relation r should be on the overall error function, one may for example set $\alpha_r = 1$ for r being the target relation, and $\alpha_{r'}$ to a small value for any auxiliary relation r'. Auxiliary relations are those relations that are included in order to learn more powerful models, usually adding these relations has a regularizing effect on the overall learning process. Finally, the choice of the loss function depends on the problem that is being addressed, for binary target

values one may use logistic loss, for continuous values usually the least squares loss is used. Additionally, for ranking problems one may choose pairwise loss functions, such as the Bayesian Personalized Ranking (BPR) loss [70].

In the following subsections, we will have a look at two relational models, firstly we will see a simple matrix factorization on one relation. Secondly, we will introduce factorization machines [69], which are quite useful when a single relation of high arity is to be learned.

5.1.1 Single Relational Matrix Factorization

For a single relational problem where the target relation is defined between two sets of entities, for instance the e-commerce example that has been mentioned, a single relational matrix factorization is usually used as in the seminal paper by Yehuda Koren [50]. This model learns an embedding $\phi : E \longrightarrow \mathbb{R}^K$ by effectively factorizing the partially observed target relation on two entities x_1 and x_2 into the product of their latent features

$$m(x_1, x_2) = \beta_0 + \beta(x_1) + \beta(x_2) + \phi(x_1)^\top \phi(x_2) \ . \tag{5.2}$$

In addition to the interaction term $\phi(x_1)^\top \phi(x_2)$, biases are also learned through the embedding $\beta : E \longrightarrow \mathbb{R}$ for each entity individually. If we now want to model users $\mathcal{U}$ and items $\mathcal{V}$ of an e-commerce system, then the set of entities for this relation is simply $E = \mathcal{U} \cup \mathcal{V}$. Let us define by Y a matrix of observations between $\mathcal{U}$ and $\mathcal{V}$, such that for a user u_i and an item v_j the entry $y_{i,j}$ is the value of the target relation. The indices $i \in \{1, ..., |\mathcal{U}|\}$ and $j \in \{1, ..., |\mathcal{V}|\}$ that we use simply adress numbers, and by u_i and v_j we adress the entities. Obviously, the entries of Y are very sparsely populated with observations, as for example in a web shop a customer usually does not give feedback to all items present in the shop, simply because the number of items is large. Conversely, there might be items that also do not receive much feedback, for instance by simply not being popular, and therefore the associated column will be highly sparse. A simple matrix factorization model attempts to find low dimensional matrices $U \in \mathbb{R}^{K \times |U|}$ and $V \in \mathbb{R}^{K \times |V|}$ such that their product, plus the bias terms, approximates the partially observed target relation Y

$$Y \approx U^\top V \tag{5.3}$$

and therefore, for any user u_i and item v_j the target relation is predicted as

$$y_{i,j} = b_0 + b_{u_i} + b_{v_j} + U_i^\top V_j \tag{5.4}$$

where the K-dimensional vectors $U_i = \phi(u_i)$ and $V_j = \phi(v_j)$ are a latent representation of the user u_i and item v_j, which are learned such that their product most accurately predicts the target relation. Obviously, the model is the same as the one in Equation 5.2. These latent features are hard to interpret, one way to think of them is that they may for example model the user's interest in a certain genre and the item's membership of the same genre. This is something they *could* be modelling, but in general, their interpretation is not possible. Having learned such a model, the target values of a pair $u_{i'}$ and $v_{j'}$ can be predicted as U and V are learned for all users and items, as long as there is at least one observed value for $u_{i'}$ and $v_{j'}$.

Usually the matrices U and V are learned by minimizing a regularized cost functional such as the least squares, at least if the target relation is continuous. In many cases, even if the target is not continuous and defined by a discrete set of levels of interest $y \in \{1,...,L\}$, the least squares can still be used, as it was done for the Netflix data for instance. Optimization is usually carried out by running stochastic gradient descent, where we go over pairs u_i and v_j iteratively and update the model.

5.1.2 Factorization Machines

Factorization machines have been proposed by Steffen Rendle in 2010 [69], and form a model class that is able to mimic many well-known models in general in machine learning and especially in recommender systems by performing a clever feature engineering. In its essence, it is a prediction model that estimates a single target value by considering all d-way interactions between the features/entities, so for a feature vector $x = (x_1,...,x_n)$ the prediction is given as:

$$m(x) = b_0 + \sum_{i=1}^{n} w_i x_i + \sum_{l=2}^{d} \sum_{t_l=1}^{n} \dots \sum_{t_l=t_{l-1}+1}^{n} \left(\prod_{j=1}^{l} x_{t_j} \right) \left(\sum_{k=1}^{K_l} \prod_{j=1}^{l} v_{t_l,f}^{(l)} \right) \quad . \tag{5.5}$$

If the feature vector for example consists of n many entities in a relational setting, a factorization machine would predict the true $f(x_1,...,x_n)$ by estimating it as a sum of all interactions up to the order d. Usually, $d = 2$ is used for factorization machines, thus making the prediction consisting of a bias term, a sum of linear terms, and a sum of all interactions of second order. For them the entity values are being multiplied, and their weight is being computed using a factorization. The prediction then reduces to

$$m(x) = b_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} v_i^\top v_j x_i x_j \quad . \tag{5.6}$$

This formulation allows to mimic several machine learning models among which are matrix factorization [50], pairwise-interaction tensor factorization (PITF) [72], Singular Value Decomposition ++ (SVD++) [49] and factorized personalized markov chains (FPMC) [71]. For example, if we want to recover the initial biased matrix factorization, we simply need to define our features x_t as

$$n = |\mathcal{U} \cup \mathcal{V}| \qquad x_t(u_i, v_j) = \chi(t = i \vee t = j) \tag{5.7}$$

which for every (u_i, v_j) pair will produce a very long sparse vector, where only two entries, the one for the respective user and the one for the respective item are nonzero. If we use this feature representation in Equation 5.6 or Equation 5.5 (as we only have two entities anyway), we exactly retrieve the prediction that is made by the biased matrix factorization. What is special about factorization machines is not only that they are able to mimic several machine learning models, but they also allow to include non-categorical features. These could be any entity features, such as the membership of a movie to a certain genre, the interest of a user for items such as drilling machines, the expected salary of the user and many more. The initial matrix factorization framework does not allow this, as there only Y is being factorized and therefore only the categorical indicator features are being used.

5.2 Relational Models for Hyperparameter Optimization

In this section, we will make use of the relational models that we have defined in the previous section, and we will illustrate how these models can be used for hyperparameter optimization. We will firstly discuss requirements on cross-data surrogate models, and then, how these requirements can be mapped to relational models.

5.2.1 Requirements for Cross-Data Surrogates

High Model Complexity Due to the response surface being usually highly nonlinear (refer to Figure 2.1 for instance), a surrogate model should be able to capture these nonlinearities in order to derive a good policy for hyperparameter optimization. As we will see in the experiments in this section, quite a few of the models that we will test to reconstruct the response surface later on fail at doing so, due to their limited complexity.

Shared and Target-Specific Parameters The goal of learning hyperparameter responses on many data sets demands for a model that can - for each data set - correctly predict the resulting validation performance on a feature vector which then only differs on a subset of features, namely the features that supposedly describe the data set. The resulting model therefore has to associate its parameters with data sets and with hyperparameters, we seek to learn models that learn individual parameters for the data set characteristics and shared parameters for the hyperparameter features.

Prediction Uncertainty If we fully trust the surrogate model m in its predictions and therefore always query the hyperparameter configuration with the best predicted performance, we are doomed to fail because only exploitation of the model is done, meaning that we always stay in a region of the hyperparameter space Λ where we have started. This is due to the fact that the surrogate model is learned on a few observations of f and therefore will not accurately predict every hyperparameter performance. As the hyperparameter optimization needs to be carried out under a Bayesian setting, we need to be able to compute uncertainties of the model, or more precisely, we need to predict a distribution over the resulting performance, instead of simply predicting an expectation value. Many machine learning models however only are interested in predicting one target value, therefore, we need to come up with ideas how to compute uncertainties.

5.2.2 Factorization Machines as Surrogates

The first model that we want to propose as surrogate model in Bayesian hyperparameter optimization is a factorization machine, specifically a factorization machine of order $d = 2$, where the prediction is computed as given in Equation 5.6. Considering the requirements on surrogate models that we have defined above, high model complexity is only given through the last term in the model's prediction, where all interaction terms are modeled between the features and their common weight is being factorized. Overall, the complexity of the model is not higher than the complexity of a polynomial regression with degree $d = 2$, which will become a problem.

Considering the shared and target specific parameters that we want to learn, a factorization machine fulfills this requirement with ease, as for each feature, we learn a linear coefficient and then an embedding to a latent space that is used to compute the interaction weights.

5.2.3 Multilayer Perceptrons as Surrogates

The second model that we want to propose is a multilayer perceptron, which is usually called a feedforward neural network or an artificial neural network. The model's prediction can be written recursively as:

$$
\begin{aligned}
m(x) &= \sigma_L(w_L + W_L h^L) \\
h^l &= \sigma_{l-1}(w_{l-1} + W_{l-1} h^{l-1}) \qquad \forall l = 2, \ldots, L \\
h^1 &= \sigma_0(w_0 + W_0 x) \qquad w_0 \in \mathbb{R}^{N_1} \quad W_0 \in \mathbb{R}^{N_1 \times d} \\
h^l &\in \mathbb{R}^{N_l} \quad w_l \in \mathbb{R}^{N_{l+1}} \quad W_l \in \mathbb{R}^{N_{l+1} \times N_l} \qquad \forall l = 1, \ldots, L
\end{aligned}
\tag{5.8}
$$

In the very first layer of the network, we use our feature vector x as input, and compute N_1 many independent sums out of it, then use σ_1 as an elementwise nonlinear activation function, which will give the input for the first hidden layer. The purpose of the activation function σ_l is to allow the model through the introduction of nonlinearities to achieve a high prediction complexity, if we would use the identity function as activation, the model would simply collapse to a linear regression. In the subsequent layers, the computations are being repeated for L many layers, where the dimensionality of each layer is given by N_l.

For the final layer, N_{L+1} has to be specified to be the size of the desired output values. Depending on the task to solve, σ_L has to be adjusted, for regression tasks one simply uses the identity function, as this allows the model to predict arbitrary real values. For classification tasks, a softmax function is usually used, and the output is interpreted as probabilites for each class. Another way to think of neural networks is that they do a supervised feature preprocessing, where in the last layer a simple linear regression or softmax classification is learned. Learning of multilayer perceptrons is usually done via backpropagation of the gradients [32]. A simple neural network with three hidden layers is given in Figure 5.1.

Given that the multilayer perceptron with L many layers is sufficient to reproduce any function if L and N_l are chosen carefully, it fulfills the first requirement that we demand from decent surrogate models. However, if we think of categorical data set indicators, the model will only associate one parameter with these indicators, for each neuron in the first hidden layer, so overall we will have N_2 many parameters associated with a data set indicator. As these parameters act as data set biases, the model actually is able to generalize across data sets, however, this will probably be very hard to achieve

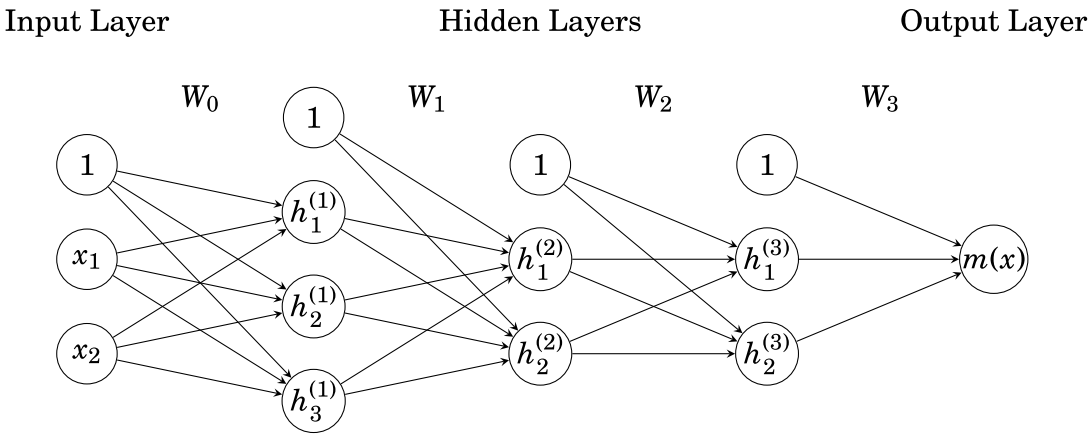

Figure 5.1: Example of a neural network structure with three hidden layers. The biases are indicated by the nodes that are constantly set to one. When the arcs meet in the neuron of the subsequent layer, their weighted sum is computed and evaluated in the activation function.

as a set of bias terms is a rather weak description of a data set. In addition to that we will not model interactions of data set entities with hyperparameter entities, as a factorization model would do.

5.2.4 Factorized Multilayer Perceptrons

A Factorized Multilayer Perceptron is the model that we propose as surrogate model in our publication [78]. It alleviates the issue of factorization machines of not offering enough model complexity, additionally it is based on factorization models and therefore learns more than only a set of data set biases which a regular multilayer perceptron would do. This is accomplished by merging factorization machines with multilayer perceptrons. Specifically, instead of feeding only a linear sum into the activation function of the first hidden layer, we add a factorization term. Then, the resulting model is given by:

$$
\begin{aligned}
m(x) &= \sigma_L(w_L + W_L h^L) \\
h^l &= \sigma_{l-1}(w_{l-1} + W_{l-1} h^{l-1}) \qquad \forall l = 2, \ldots, L \\
h^1 &= \sigma_0\left(w_0 + W_0 x + \left((Vx)^2 + V^2 x^2\right)\mathbb{1}\right) \\
w_0 &\in \mathbb{R}^{N_1} \quad W_0 \in \mathbb{R}^{N_1 \times d} \quad V \in \mathbb{R}^{N_1 \times k \times d} \quad \mathbb{1} \in \mathbb{R}^k \\
h^l &\in \mathbb{R}^{N_l} \quad w_l \in \mathbb{R}^{N_{l+1}} \quad W_l \in \mathbb{R}^{N_{l+1} \times N_l}
\end{aligned} \tag{5.9}
$$

The main difference to the multilayer perceptron is the way that the first hidden layer h^1 is computed. Let us denote by V the tensor that holds for N_1 many output neurons and k many latent features for each of the d many features of the input vector. Then, the multiplication in

$$h^1 = \sigma_0\left(w_0 + W_0 x + \left((Vx)^2 + V^2 x^2\right)\mathbb{1}\right) \tag{5.10}$$

of V and x sums over the d many dimensions of x. The term in the bracket is then summed up over its latent features, which is done by multiplication with a vector $\mathbb{1}$ that is filled with ones. In addition to that, the square operation is supposed to be understood elementwise, so for example

$$x^2 = \left(x_i^2\right)_{i=1}^d. \tag{5.11}$$

Algorithm 2 SGD-Backpropagation for Factorized Multilayer Perceptron

Require: Data Set D, Loss function ℓ, step length $\eta > 0$.

1: **repeat**
2: Draw $(x, y) \in D$
3: Predict $m(x)$
4: Compute $\delta_k^l = \frac{\partial \ell(m(x),y)}{\partial s_k^l} \cdot \frac{\mathrm{d}\sigma}{\mathrm{d}s_k^l}$ for all layers l and nodes k
5: Update $(W_l)_{i,k} = (W_l)_{i,k} - \eta \delta_k^l x_i$
6: Precompute $\mu_j^k = \sum_{i=1}^n v_{i,j}^k x_i$
7: Update $v_{i,j}^k = v_{i,j}^k - \eta \delta_k^l (x_i \mu_j^k - v_{i,j}^k x_i^2)$
8: **until** Convergence

By adding factorization machines to the input layer, we effectively model all feature interactions in x and associate k many latent features with the input feature in order to compute the interaction weights. This way, we have designed a model that fulfills

both the complexity requirement as well as the requirement of learning target specific parameters in a more meaningful way than simply having bias terms. However, all of the proposed surrogate models so far lack the ability of computing prediction uncertainties, which was the third requirement on surrogates.

FMLP Extensions

So far, and in our publication in [78] we have evaluated the FMLP model that was proposed above, however, a few extensions of it may be considered. At first, we observe that we only in the first layer use factorization machines as signal for the activation function. This has two main reasons. We initially used a factorization model in order to learn latent characteristics of different problem instances in hyperparameter optimization such as different data sets. As the data set information is only given in the input layer via a one-hot encoding, using a factorization model only makes sense in the input layer, as the information of the dataset is diffused through the hidden layers. In addition to that, we believe that the complexity of a neural network is high enough, so that additional factorizations in subsequent layers are not needed.

5.2.5 Estimating Predictive Posteriors for MLP and FMLP

As stated above, both MLP and FMLP are currently unable to be applied to sequential model-based optimization, as the acquisition function demands a distribution over hyperparameter performance, instead of a single maximum likelihood estimate only. In this section, we will shortly describe Bayesian Neural Networks, as they are presented in [64, pp. 579-580] and more detailed in [8, pp. 277-280]. By treating the proposed surrogate models in a Bayesian setting, it is possible to deduce prediction uncertainty using various ways, such as (hybrid) Monte Carlo methods as they are presented in [65] or by variational Bayes as in [36] for example. We follow the approach of Laplace approximation which was first presented by MacKay in [58].

Laplace Approximation of the Predictive Posterior

The Laplace approximation works by estimating a distribution $p(z)$ that is defined over continuous variables. The approximation is done by estimating a Gaussian $q(z)$, which is computed from the second order Taylor series of $\ln p(z)$ which is expanded around a mode z_0 of $\ln p(z)$. So let us assume we have found by some numerical optimization a

mode z_0, so

$$\nabla_z p(z)|_{z=z_0} = 0 \ . \tag{5.12}$$

Now, expanding the Taylor series around z_0 and cutting it off after order two, we obtain

$$\begin{aligned} \ln p(z) &= \sum_{|\alpha|\geq 0} \frac{(z-z_0)^\alpha}{\alpha!} \partial^\alpha \ln p(z_0) \\ W \qquad &\approx \ln p(z_0) + \nabla_z \ln p(z_0)(z-z_0) - \frac{1}{2}(z-z_0)^\top \nabla_z \nabla_z \ln p(z_0)(z-z_0) \\ &= \ln p(z_0) - \frac{1}{2}(z-z_0)^\top \nabla_z \nabla_z \ln p(z_0)(z-z_0) \end{aligned} \tag{5.13}$$

where the first order term disappears as the gradient at the mode is equal to zero. Exponentiating the terms on both sides delivers that

$$\begin{aligned} p(z) &= p(z_0) \exp\left(-\frac{1}{2}(z-z_0)^\top \nabla_z \nabla_z \ln p(z_0)(z-z_0)\right) \\ &\propto \mathcal{N}(z \,|\, z_0, A^{-1}) \end{aligned} \tag{5.14}$$

where equality is not given since we do not consider normalization terms. Additionally, we denote

$$A = -\frac{1}{2}(z-z_0)^\top \nabla_z \nabla_z \ln p(z_0)(z-z_0) \ . \tag{5.15}$$

This means that we need to compute the Hessian matrix of $\ln p(z)$ at $z = z_0$, to conduct the Laplace approximation.

Now we want to use the Laplace approximation to compute the predictive posterior of both proposed surrogates, the MLP and the FMLP. Let us therefore denote by Θ a single variable containing all parameters of m, including biases, weights and possibly latent characteristics, so for an MLP it would contain

$$\Theta_{\text{MLP}} = \left(\{w_l\}_{l=0}^L \ \{W_l\}_{l=0}^L\right) \tag{5.16}$$

or alternatively for an FMLP

$$\Theta_{\text{FMLP}} = \left(\{w_l\}_{l=0}^L \ \{W_l\}_{l=0}^L \ V\right) \tag{5.17}$$

as the latent feature tensor V would also be included. We can think of Θ as a flattened vector that contains all model parameters, as we will see, Θ will play the role of z in the abovely presented Laplace approximation. The first thing we need to do then is define a

prior distribution $p(\Theta \mid \alpha)$, where we will assume a Gaussian prior with covariance α^{-1} on Θ of the form

$$p(\Theta \mid \alpha) = \mathcal{N}\left(\Theta \mid \mathbf{0}, \alpha^{-1}\mathbf{I}\right) \tag{5.18}$$

where $\mathbf{I}$ is a d dimensional identity matrix where we assign $d = \dim(\Theta)$. The density then has the form

$$\begin{aligned} p(\Theta \mid \alpha) &= \frac{1}{\sqrt{(2\pi)^d \det(\alpha^{-1}\mathbf{I})}} \exp\left(-\frac{1}{2}(\Theta - \mathbf{0})^\top (\alpha^{-1}\mathbf{I})^{-1}(\Theta - \mathbf{0})\right) \\ &= \frac{1}{\sqrt{(2\pi)^d \alpha^{-d}}} \exp\left(-\frac{\alpha}{2}\Theta^\top \Theta\right) \quad . \end{aligned} \tag{5.19}$$

For the likelihood of observing one instance (x, f) given our model parameters Θ and assuming that $f \in \mathbb{R}$ as in a regression problem, we assign a Gaussian likelihood of the form

$$\begin{aligned} p(f \mid x, \Theta, \beta) &= \mathcal{N}(f \mid m(x, \Theta), \beta^{-1}) \\ &= \frac{1}{\sqrt{2\pi\beta^{-1}}} \exp\left(-\frac{\beta}{2}(f - m(x, \Theta))^2\right) \end{aligned} \tag{5.20}$$

where β is the precision of the likelihood or can also be understood as the inverse of the data set noise. As usual, we assume that our data samples are identically independently distributed, so the likelihood of the whole data set can be written as

$$\begin{aligned} p(D \mid \Theta, \beta) &= \prod_{i=1}^{|D|} \mathcal{N}\left(f_i \mid m(x_i, \Theta), \beta^{-1}\right) \\ &= \prod_{i=1}^{|D|} \frac{1}{\sqrt{2\pi\beta^{-1}}} \exp\left(-\frac{\beta}{2}(f_i - m(x_i, \Theta))^2\right) \quad . \end{aligned} \tag{5.21}$$

Combining the parameter prior and the data likelihood we can estimate the parameter posterior using Bayes rule

$$p(\Theta \mid D, \alpha, \beta) \propto p(\Theta \mid \alpha) p(D \mid \Theta, \beta) \tag{5.22}$$

As Bishop argues, this is not a Gaussian due to the nonlinear influence that is introduced

by m and its parameters Θ on x. We can take the logarithm of the posterior to obtain

(5.23)
$$\begin{aligned}\ln(p(\Theta \mid D, \alpha, \beta)) &\propto \ln(p(\Theta \mid \alpha)) + \ln(p(D \mid \Theta, \beta)) \\ &= \ln(\frac{1}{\sqrt{(2\pi)^p \alpha^{-p}}}) + \frac{\alpha}{2}\Theta^\top\Theta + \sum_{i=1}^{|D|}\left(\ln\left(\frac{1}{\sqrt{2\pi\beta^{-1}}}\right) + -\frac{\beta}{2}(f_i - m(x_i,\Theta))^2\right) \\ &= C - \frac{\alpha}{2}\Theta^\top\Theta - \sum_{i=1}^{|D|}\frac{\beta}{2}(f_i - m(x_i,\Theta))^2\end{aligned}$$

where we have collected all terms that do not depend on Θ in the constant C.

This is the point where the Laplace approximation comes into play. At first, we need to find a mode of the posterior $\ln p(\Theta \mid D)$, while assuming that α and β are fixed. As we can see in Equation 5.23, the task is similar to minimizing a regularized sum of squares, where we can use optimization techniques such as gradient descent. Out of this reason, we call the mode the maximum a posteriori parameter vector Θ_{MAP}. Following the steps of the Laplace approximation, we therefore need to compute the Hessian matrix of 5.23, so we obtain

(5.24)
$$\begin{aligned}\nabla_\Theta\nabla_\Theta \ln(p(\Theta \mid D) &= \nabla_\Theta\nabla_\Theta\left(-\frac{\alpha}{2}\Theta^\top\Theta - \sum_{i=1}^{|D|}\frac{\beta}{2}(f_i - m(x_i,\Theta))^2\right) \\ &= \alpha\mathbf{I} + \beta H\end{aligned}$$

where H is defined as the Hessian matrix of the loss without regularization. Then, from the Laplace approximation, we can approximate

(5.25)
$$p(\Theta \mid D, \alpha, \beta) \approx \mathcal{N}(\Theta \mid \Theta_{\text{MAP}}, A^{-1}) \ .$$

Marginalizing over the model parameters Θ, the densitiy of the predictive posterior can then be written as

(5.26)
$$p(f \mid x, D, \alpha, \beta) = \int \mathcal{N}(f \mid m(x,\Theta), \sigma^2)\mathcal{N}(\Theta \mid \Theta_{\text{MAP}}, A^{-1})\mathrm{d}\Theta \ .$$

As [64] and [8] argue, this integral is not feasible to compute because of the nonlinearity of m, thus we will have to conduct another Taylor approximation on m to make it linear. Expanding m around Θ_{MAP} yields

(5.27)
$$m(x,\Theta) \approx m(x,\Theta_{\text{MAP}}) + \nabla_\Theta m(x,\Theta)|^\top_{\Theta=\Theta_{\text{MAP}}}(\Theta - \Theta_{\text{MAP}}) \ .$$

Now, having a Gaussian prior

$$p(\Theta) = \mathcal{N}(\Theta \mid \Theta_{\text{MAP}}, A^{-1}) \tag{5.28}$$

for our model parameters, and linear Gaussian model for our likelihood

$$p(f \mid \Theta) = \mathcal{N}(m(x, \Theta_{\text{MAP}}) + \nabla_\Theta m(x, \Theta)|^\top_{\Theta=\Theta_{\text{MAP}}} (\Theta - \Theta_{\text{MAP}}), \sigma^2) \tag{5.29}$$

we can compute the marginal $p(f)$ by using the result about Gaussian conditionals from subsection A.1.2 from appendix A. Finally, since we are having a Gaussian linear model [8], the predictive posterior can be written as Gaussian

$$p(f \mid x, D, \alpha, \beta) = \mathcal{N}(f \mid m(x, \Theta_{\text{MAP}}), \beta^{-1} + g^\top A^{-1} g) \tag{5.30}$$

or analogously

$$\begin{aligned} \mu(f) &= m(x, \Theta_{\text{MAP}}) \\ \sigma^2(f) &= \beta^{-1} + g^\top A^{-1} g \end{aligned} \tag{5.31}$$

where we have defined

$$g = \nabla_\Theta m(x, \Theta)|_{\Theta=\Theta_{\text{MAP}}} \quad . \tag{5.32}$$

In conclusion, to predict the uncertainty of m for an instance x, we need to estimate the Hessian of the loss of m on D, and a gradient g depending on x. The latter task is easy at it only involves a computation of the gradient, which is for a multilayer perceptron a forward and a backward pass through the network.

To compute the inverse Hessian in an analytic fashion is usually not feasible as computing only one entry of the Hessian involves a pass over the whole data and then inverting the resulting matrix has an effort that is cubical in the number of parameters, i.e. the dimensionality of Θ. Out of this reason, we seek to approximate the inverse of the Hessian directly by using a sum of outer products as it is exposed in [8]. As the target loss is least squares, the Hessian can be written as

$$H = \sum_{(x,f)\in D} \nabla m(x, \Theta) \nabla m(x, \Theta)^\top + \sum_{(x,f)\in D} (f - m(x, \Theta)) \nabla \nabla m(x, \Theta) \quad . \tag{5.33}$$

As [8] outlines, for a carefully learned model the second sum can be neglected as the quantity $(f - m(x, \Theta))$ is close to zero. Thus, H can be approximated using only the first

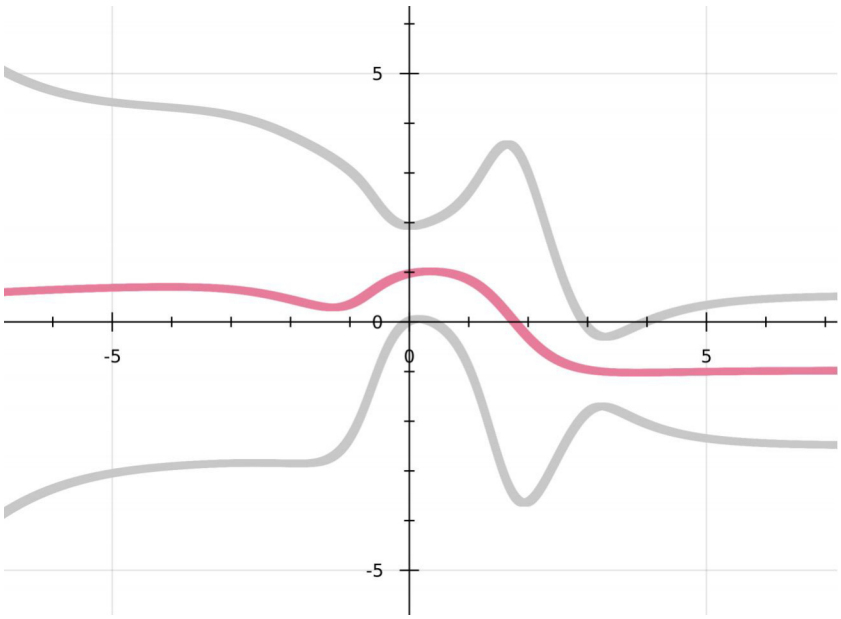

Figure 5.2: Predictive Posterior of a multilayer perceptron learned on $(x_1, y_1) = (0, 1)$ and $(x_2, y_2) = (\pi, -1)$. The red line shows the mean, the grey line shows one standard deviation (Best viewed in color)

term which is a sum of outer products. The inverse of H can directly be computed in an iterative fashion, where we initialize the Hessian as

$$H^{-1} = \alpha^{-1} I \tag{5.34}$$

and then go over the data set. Every instance we consider will add a rank-one update to H, so we can compute the inverse after each instance using the Sherman-Woodbury formula

$$\left(H + \nabla m(x, \Theta) \nabla m(x, \Theta)^\top\right)^{-1} = H - \frac{\nabla m(x, \Theta) \nabla m(x, \Theta)^\top}{1 + \nabla m(x, \Theta)^\top \nabla m(x, \Theta)} \ . \tag{5.35}$$

In this way, we we effectively compute the inverse of $H + \alpha I$, which is exactly the matrix we seeked to invert.

A one dimensional example can be seen in Figure 5.2, where a multilayer perceptron is learned on two data points. As we can see, the uncertainty that is computed makes sense, however, there is an apparent drawback that is visible in the plot. Depending on the data set noise β^{-1}, the resulting uncertainty is still too large around points that have already been observed. To decrease the uncertainty at these points, we need more data samples and in hyperparameter optimization, these samples are very costly to compute. Another disadvantage of the Laplace approximation is that the approximation is only

done locally around one mode, but global knowledge of the posterior is completely left out, which may also lead to the performance issues as we will see in the experiments.

Estimating Uncertainty from Ensembles

As an alternative approach, we consider computing an ensemble of K many surrogates and predict the uncertainty using the estimated mean and standard deviation of all the predictions, as it is also done by other surrogate models such as SMAC. Formally, we compute the surrogate's mean and variance using

$$\begin{aligned} \mu(f) &= \frac{1}{K} \sum_{i=1}^{K} m_i(x) \\ \sigma^2(f) &= \frac{1}{K-1} \sum_{i=1}^{K} (m_i(x) - \mu(f))^2 \quad . \end{aligned} \tag{5.36}$$

The resulting variance stems only from different initializations of the surrogate model, which is reasonable if the whole optimization problem is highly non-convex and therefore yields different solutions. This even is an advantage over the Laplace approximation as this approach also finds different modes, and therefore the gathered knowledge about the posterior is more globally accurate. Compared to the Laplace approximation, we have an additional effort in learning all these different models, but we do not need to compute the Hessian across the whole data set.

Estimating Uncertainty from the Parameter Sequence

Another way of approximating the posterior around a mode can simply be done by sampling different model parameters when the mode is being approached. As every optimization step usually jumps around the mode, we could sample the model parameters after every K-th iteration, and then compute the surrogate's mean and uncertainty using the formulas above. However, this approach has its limitations. At first, compared to the approach of learning ensembles, this approach only approximates the posterior around one mode and therefore may not have global knowledge. Secondly, we have to specify K in a meaningful way, as we seek to not have the same model sampled many times. Unfortunately, K also depends on the learning rate of the optimization algorithm. Lastly, deciding when we have approached a useful mode to begin sampling is a tough question as we may get stuck in a bad mode. Out of these reasons, we only compare the first two approaches in the experiments in this chapter.

5.3 Empirical Evaluation

To assess the performance of our proposed surrogate models we will conduct three different experiments on the SVM and the AdaBoost meta data sets. A detailed description of how these meta data sets have been created as well as the evaluation metrics that we employ to asses performance can be found in chapter 4. The first experiment deals with how well the proposed surrogates can reconstruct the response surface, so we are basically only solving a regression problem. The second experiment compares the two different ways of estimating prediction uncertainty for a multilayer perceptron. Finally, the third experiment assesses the performance when predicting hyperparameters to choose in the sequential model-based optimization setting.

5.3.1 Competing Surrogate and Regression Models

In this part, we give a short overview of the surrogate and regression models that were evaluated in all of the experiments in [78] and link them to their more detailed description in chapter 3. Since we evaluate different tasks - for instance only want to model the response surface using a regression model - not all of the models are evaluated in all of the experiments.

Random Search Simply sampling hyperparameters from a distribution instead of a grid-like fashion was proposed by [7]. Random search does not use knowledge from other data sets, additionally it does not even use the information it has gathered on the target data set in any way.

Independent Gaussian Process (I-GP) We call the approach by [86] an independent Gaussian process, it is independent as it does not learn information across data sets.

Random Forest (RF) Using a random forest as a surrogate was proposed by Frank Hutter in [39], this model learns an ensemble of regression trees and uses them to estimate a mean and standard deviation for the performance of each hyperparameter configuration. For the implementation, we used the MLTK library [55].

Surrogate-based Collaborative Tuning (SCOT) The work by [3] uses a ranking SVM and learns a Gaussian process on the ranker's output, to estimate uncertainties.

Support Vector Regression (SVR) In analogy to the SVM based model that is used by Bardenet et al. [3], we evaluate a support vector regression on our meta data. We used the famous SVM Light library from [40].

Factorization Machine Regression (FM) We use a simple factorization machine as was proposed by Rendle in [69], which is a polynomial model where the interaction weights are factorized.

Multilayer Perceptron (MLP) and Factorized Multilayer Perceptrons (FMLP) The surrogate models presented in this chapter and in [78]. We have implemented these models in Java by ourselves.

Optimal An artificial tuning strategy that always picks the best working hyperparameter configuration in the first trial. We use it only for orientation purposes.

5.3.2 Reconstruction of Response Surfaces

The first requirement we postulated was high model complexity; in this experiment we are going to compare the complexity of the proposed models and how well they can regress on the response surfaces. The experiment is designed in a leave-one-out fashion, where we learn the surrogate on $|\mathcal{D}| - 1$ many response surfaces and a small percentage of observed hyperparameter configuration for the test data set. Specifically, we used 4% of the test data as additional training (probing) data, 10% as validation data for hyperparameter optimization of the regression models, and the remaining 86% as test data. All models are optimized to achieve a minimal Root Mean Squared Error (RMSE). The hyperparameters have been optimized using grid search, the employed grid of hyperparameters can be found in [78].

Table 5.1: Confidence intervals of the resulting RMSE for all models when reconstructing the response surface of SVM and AdaBoost

	RF	SVR	FM	MLP	FMLP
SVM	0.0997 ± 0.028	0.1110 ± 0.020	0.1041 ± 0.029	0.0596 ± 0.013	$\mathbf{0.0550} \pm 0.016$
AdaBoost	0.0462 ± 0.012	0.0840 ± 0.009	0.0579 ± 0.015	0.0380 ± 0.008	$\mathbf{0.0377} \pm 0.009$

The RMSE values and their respective 95% confidence intervals are shown in Table 5.1. At first, we can clearly observe that both the proposed models - MLP and FMLP -

perform well in reconstructing the response surface, as their RMSE is quite low. The FMLP tends to achieve better RMSE values than MLP; however, the improvement is marginal and statistically not significant. Random Forests also seem to work well in regressing on the hyperparameter performance, as on AdaBoost the difference to both FMLP and MLP is also not statistically significant. The support vector regression is overall the worst model, followed by factorization machines as the second worst.

At first, the bad performance of the factorization machine is a little surprising. This stems from the fact that its expressivity is rather limited for both meta data sets. The limitedness of an FM can be shown by an easy example. Let us consider an instance out of the SVM meta data where we have chosen the RBF Kernel, a cost term C and and a Kernel width γ, then the FM prediction has the form

$$m(x) = w_0 + w_C C + w_\gamma \gamma + w_{C,\gamma} C\gamma \qquad x = (C, \gamma) \tag{5.37}$$

where we omit the meta features of the data set, as for one target data set, these are constant and will only be multiplied with C and γ. Therefore the above term can be understood in a way that the metafeature influence is already in the respective weights. The geometrical form of the prediction is a hyperbolic paraboloid and therefore quite limited in predicting the shape of response surfaces. Out of this reason, we do not consider the factorization machine any longer as a potential surrogate model, due to its limited expressivity.

5.3.3 Uncertainty Estimation

To compare different ways of estimating uncertainties of our proposed surrogate models, we performed an experiment using MLP as a surrogate model in two variants that are presented in section 5.2.5. The first variant is called MLPH and uses the Laplace approximation in order to compute its uncertainty. The second variant uses an ensemble of MLPs to estimate uncertainties, where we use 100 individual MLP models in the ensemble, we will denote this by MLPE. Experiments are run in a leave-one-data-set-out fashion, meaning that for example for AdaBoost, we learn the surrogate model on the meta observations of all data sets except for the target data set, repeat this as many times such that each data set has been the target data set once, and finally average the results across all runs. Figure 5.3 shows the development of the average hyperparameter rank on both the AdaBoost and the SVM meta data set.

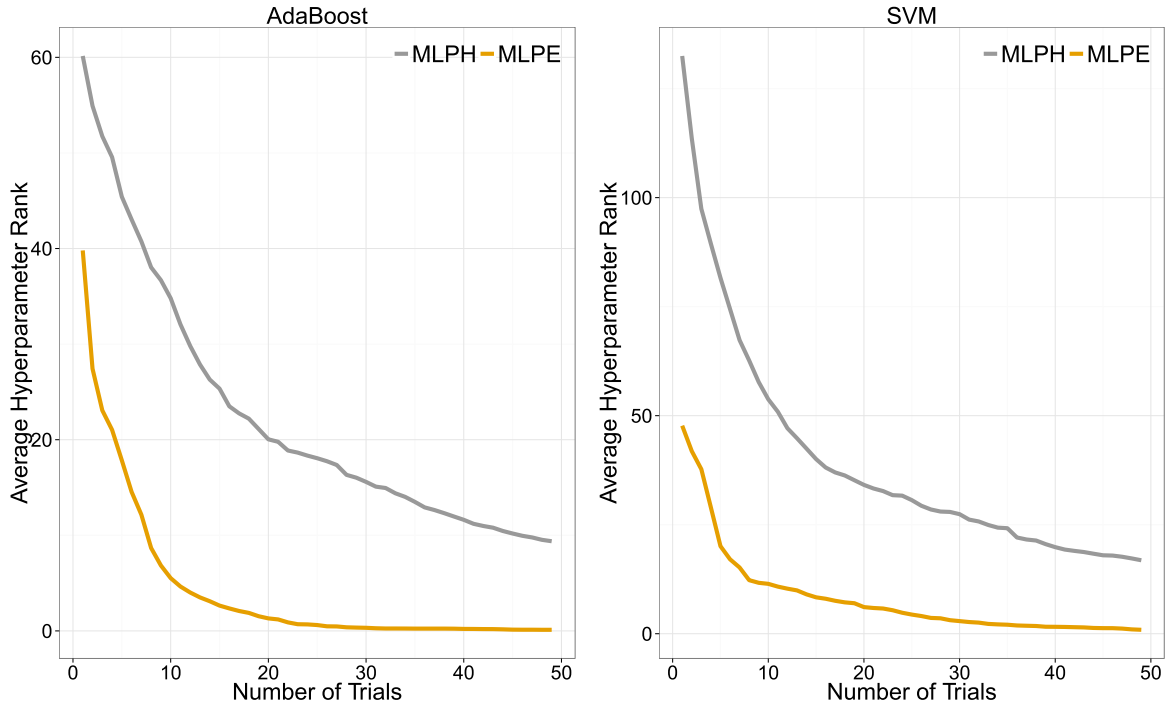

Figure 5.3: Development of the average hyperparameter rank with increasing numbers of trials. Clearly, the convergence of the ensemble MLP is much faster than using the inverse Hessian (Best viewed in color)

Results are averaged over ten runs, to cancel out random effects from different model initializations. As both plots indicate, the hyperparameter configurations chosen by MLPE are much better, which is due to several reasons. First of all, the Laplace approximation of the posterior contains many assumptions and simplifications, such as the Taylor approximation and the neglection of the second sum in Equation 5.33. This term only vanishes for a well-learned model, however, in SMBO we sequentially add information of the new data set, for which the surrogate cannot be optimally learned in practice. In addition, it seems that many observations are needed to obtain tight error bars, i.e. error bars that disappear close to observed points. However, already in the one-dimensional example in Figure 5.2 this does not work too well, at least without more observations, which are extremely costly to compute in our application. The effect of relatively high uncertainty around already observed hyperparameter configurations is crucial, as the acquisition function may start to do exploitation. This may occur, as for a decent working configuration, the uncertainty is still high, and therefore searching in the vicinity of this decent configuration does not lead to a huge improvement, in contrast to what the acquisition function would suspect.

Having conducted this experiment, we decided to no longer use the uncertainty estimation using the Laplace approximation. Instead, we learn individual ensembles only for the upcoming experiments.

5.3.4 SMBO-based Hyperparameter Optimization

Finally, we want to asses how well our proposed surrogate models work in Bayesian hyperparameter optimization, specifically in sequential model-based optimization. We perform a similar experiment as the one above, this time however we compare different surrogates and not the same surrogate using different ways of estimating uncertainty. This means, we again conduct the leave-one-data-set-out scheme that was discussed in the previous subsection. Again, to account for random effects and initialization variance of model parameters, we perform every experiment 10 times and average over the results. For the random surrogate, we even conducted 1,000 different runs.

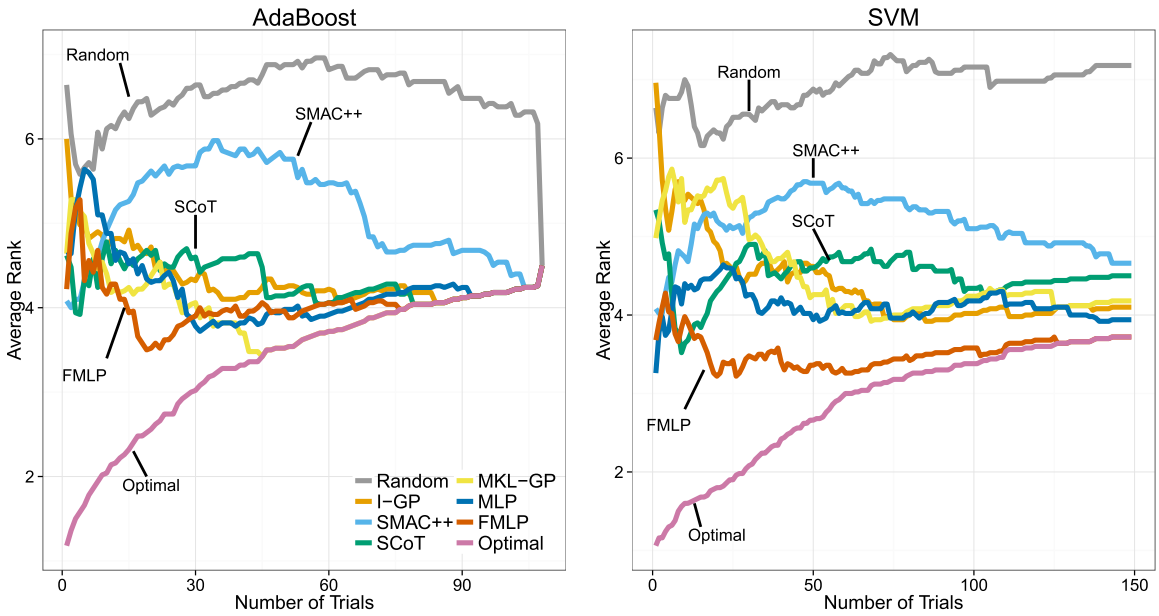

Figure 5.4: Development of the average rank for AdaBoost (left) and the SVM (right) meta data set over an increasing numbers of trials.

Figure 5.4 shows the development of the average rank for AdaBoost on the left and for SVM on the right side of the figure. At first, we see that random clearly behaves worst, whereas the optimal strategy shows the best average rank, demonstrating that the performance metric is calibrated correctly. In the beginning of the SMBO iterations, so roughly in the first ten iterations, we see that the performance of all surrogates is

still quite noisy. This is due to the fact that for the target data set there have been no observations made yet. The average rank of FMLP is given in light brown and, the average rank of MLP is given in dark blue. In opposition to the experiment results in the response surface reconstruction, we see a clear difference between the performance of FMLP and MLP, where FMLP is performing much better, this is especially visible on SVM, where FMLP outperforms all other methods clearly. We are surprised that SMAC++ performs poorly compared to all other surrogates, despite having the best starting configuration on AdaBoost. Nevertheless, its performance is underwhelming as after a few trials it even performs worse compared to models which do not learn across data sets, such as the independent Gaussian process. SCOT and MKL-GP perform reasonably well overall.

Another view on the results is offered by Figure 5.5, where the average hyperparameter rank is plotted. FMLP again shows the best performance overall, as it curve is the lowest. On the SVM meta data, it also shows the best starting configuration, having an average hyperparameter rank of almost 50, where on AdaBoost SMAC++ still has the best initializiation, but quickly degrades in performance. A little bit surprising is the average hyperparameter rank of MKL-GP, as it does not seem to perform as well with respect to this evaluation measure. This may be due to the fact that MKL-GP achieves a comparable accuracy that is only slightly worse but may have a much higher rank, maybe because the response surface has a large plateau. In this case, for the average hyperparameter rank this gets punished a lot.

Now the question we want to answer is why FMLP outperforms the other methods. We argue that the better result of FMLP stems from the explicit modelling of latent characteristics, which seem to be more useful than simply having a data set bias. On AdaBoost, MKL-GP is the first one to meet the average rank of the optimal strategy, however, we believe that AdaBoost is a rather simple meta data set as it involves only two hyperparameters. This assumption is also supported by the fact that many surrogate models work well on AdaBoost.

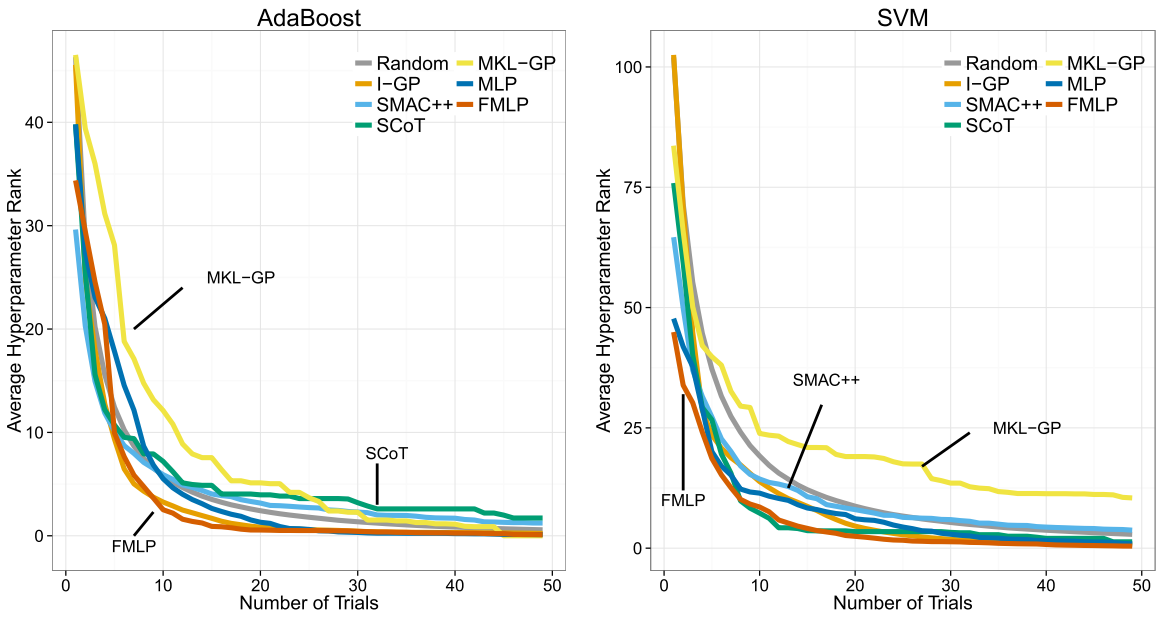

Figure 5.5: Development of the average hyperparameter rank for AdaBoost (left) and the SVM (right) meta data set over an increasing numbers of trials.

By contrast, the SVM meta data already involves some form of model choice and hierarchical hyperparameters, which are both injected into the problem through the Kernel choice. In this case, FMLP seems to have an advantage as the factorization learns interactions between hyperparameters where it needs to be modelled and learns a zero interaction, where it makes sense, for example between the degree of the polynomial kernel and the choice of the Gaussian Kernel. As both things are never observed together, their interaction $x_i x_j$ is always zero as one of the terms x_i or x_j is zero. Therefore all the latent characteristics v_i and v_j that specify the interaction weights are not updated, which only makes sense. For other models, using zeros for hyperparameter configurations that are not active may introduce a bias term that confuses the surrogate models, however, as not many methods can deal properly with missing values, there is no other direct way of setting these values correctly for all surrogates.

We believe that these experiments show that using methods from relational learning such as factorization models is quite useful for Bayesian hyperparameter optimization. However, using factorization machines directly does not seem to work, due to the limited complexity of the model. Therefore, we joined both factorization models and highly nonlinear models such as multilayer perceptrons, to come up with a model that is capable of reconstructing a highly nonlinear response surface, while still able to learn latent characteristics of categorical inputs, such as a data set indicators.

CHAPTER 6

Scalable Hyperparameter Optimization with Gaussian Process Ensembles

With more and more machine learning experiments being conducted every day, naturally the amount of meta knowledge is growing every day, given that these experiments are done in a reproducable manner or the results are being stored. Machine Learning encompasses areas such as image classification and object detection for autonomous driving, time-series analysis for sensor data of the economy 4.0, relational learning in e-commerce, medical applications and many more where data is being gathered on a daily basis. For all these problems machine learning models are learned while well-performing hyperparameters are found, in fact all of these experiments could be used to steer hyperparameter optimization for the future. One may also think of the millions of experiments in reinforcement learning scenarios such as learning how to play more and more complex video games, as simulation of these games is easily possible. In addition, learning on stream-based data seems to be an area where a lot of meta knowledge can be gathered, at least if the stream has enough volume and on the frequency of models being relearned and evaluated is high. Both of these areas also have another interesting aspect, the meta knowledge collected would be time dependent. For reinforcement learning it is time dependent through the policy that is acting better, in stream based data simply through the sequential nature of the measurements.

Unfortunately, getting access to such meta data is usually not possible due to privacy

and economical reasons in many areas. However, there are existing approaches to share meta knowledge across the community such as `www.openml.org` which is also described in detail in [92]. By end of April 2018, OpenML consists of roughly over 9 million individual experiments. This sounds large, but given that our Weka meta data set presented earlier already consists of 1.3 million experiments, the size sounds manageable.

Nevertheless, already these sizes of meta data sets can become a problem for surrogate models to handle. Given that meta knowledge is generally increasing and can be used for many hyperparameter optimization tasks, we need to design surrogate models that perform well on predicting hyperparameter configurations for new problems. It is generally agreed that Gaussian processes form a decent surrogate model as they are fully probabilistic and therefore give access to their prediction's uncertainty. Another advantage that we will see, is that their hyperparameters can be estimated by a straightforward gradient optimization maximizing the marginal likelihood of the kernel hyperparameters. Gaussian processes have a substantial downside, as for both prediction and optimization of the kernel hyperparemeters the kernel covariance matrix which is square in the size of the training data has to be inverted. In the aforementioned scenario of having large amounts of meta knowledge available, this downside is quite challenging to deal with.

There are methods on making Gaussian processes scalable to larger amounts of data but most of these approaches rely on learning on a subset of data called the active set. Most methods then sequentially add new data points to the model, while trying to carefully choose new instances, either by clustering instances first and then picking from different centers or by other means such as greedy selection as in [56] and [85]. For the purpose of hyperparameter optimization, these sparse approaches are not really applicable, as we already have a sequential decision problem of picking new hyperparameter configurations on the test problem and therefore want to include as much meta knowledge as possible especially in the beginning. Starting out SMBO with a surrogate that only includes a small percentage will then lead also to poor decisions in the beginning which then likely negatively influence the whole search.

In order to maintain the idea of using GPs as surrogates as well as making use of the whole amount of meta knowledge, we demonstrate that models based on products of experts [10] - in our case Gaussian process experts - work well for hyperparameter optimization. We also investigate Bayesian committee machines (BCM) as presented in [91], which offer a different weighting scheme compared to product of experts models. We

will demonstrate that these approaches, as well as new approaches derived from them show convincing performance.

In this chapter, we will present the results of several publications. At first, we present the results of [80], where scalable surrogate models are developed using products of Gaussian process experts. Secondly, we shortly present the work of Martin Wistuba in [94], as the proposed surrogate models are quite similar, but differ in some aspects. Finally, we present the contents of [95], where both of the aforementioned publications have been merged to a joint paper which has been published in the Machine Learning journal. The final work also expands on the individual works, by proposing not only transfer surrogate models but also transfer acquisition functions, which show even stronger performance. Instead of evaluating the acquisition function for each surrogate in the ensemble individually, all models are evaluated jointly for transfer acquisition functions. We will start the chapter by introducing Gaussian process as they are the base model used throughout this chapter.

6.1 Gaussian Process Regression for Noisy Data

Arguably one of the most commonly used stochastic process besides others such as random walk and Wiener process is probably a Gaussian process. In this section, we will introduce Gaussian processes, their advantages and disadvantages, and how to make inference with the model, all of these things are described well in [68] for example.

6.1.1 Gaussian Process Priors

As many other regression models, Gaussian processes consider a model of the form

$$y(x_i) = f(x_i) + \epsilon \qquad x_i \in \mathcal{X} \tag{6.1}$$

where $f(x_i)$ is the underlying function that we seek to estimate, we will also write $f_{x_i} = f(x_i)$ sometimes. Unfortunately, the function f is distorted by a homoscedastic noise $\epsilon \sim \mathcal{N}(0, \sigma_y^2)$ that is independent of x. Assuming such a prior noise exists is quite reasonable for our scenario, as many random effects that occur in training machine learning models cannot entirely be captured by the hyperparameter configuration which is contained in x. To discuss some examples, the first thing that comes to mind is the way the learning algorithm makes use of the data, for example the sequence in which data instances are used in stochastic gradient descent. This does not only influence the

sequence in which model parameters are learned but also has an effect on stopping criteria such as using the gradient norm. Another stopping criterion is early stopping on validation data, which inherently has a random effect from the data distribution between training and validation data. Additionally, the random initialization of model parameters which often follows a Gaussian distribution may have a high influence on the final result. This is especially true for non-convex problems for example when training deep learning models for data cubes. Arguably, one could say the only hyperparameter to control this randomness is the variance of the parameter initialization, making the problem somewhat heteroscedastic, as for example high values of initial variance usually do not work well in practice. However, for highly non-convex problems, knowing only the variance does not explain much of the randomness in learning, particularly if a good variance has already been configured.

In order to estimate the unknown function f, we will have to assume a Gaussian process on the observed performances y in our meta data. A Gaussian process is a functional representation of a possibly infinite dimensional multivariate Gaussian. Given that we have an infinite set of observations y, GPs can be handled with the calculus of multivariate Gaussians which is presented in the appendix of this thesis.

Definition 6.1. We say that a set of random variables $f_x \in \mathcal{F}$ follows a Gaussian process if for all finite subsets of $\mathcal{X}$, the finite vector of f_x observations follows a multivariate Gaussian with mean μ and covariance K

$$\forall x_1, ..., x_n : \quad f_{x_1}, ..., f_{x_n} \sim \mathcal{N}(\mu, K) \ . \tag{6.2}$$

In some books, the following notation is also used

$$(f_x) \sim \mathcal{GP}\left(\mu(x), k(x, x')\right) \ . \tag{6.3}$$

A Gaussian process is a stochastic process where every finite sample of observed values follows a multivariate Gaussian distribution whose mean μ is specified by a mean function $\mu(x)$ and the covariance K being determined by a symmetric positive definite kernel function

$$k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}^+ \tag{6.4}$$

such that $k(x, x') = k(x', x)$ and

$$\sum_i \sum_j c_i c_j k(x_i, x_j) \geq 0 \tag{6.5}$$

for all $x \in \mathcal{X}$ and $c_i, c_j \in \mathbb{R}$.

The probability density of a Gaussian process has no analytical form, however, as we already mentioned in real-life applications we only ever observe hyperparameter performance for a finite set of configurations Therefore, using the Gaussian process for inference only ever requires the well-known calculus of multivariate Gaussians as we will see soon.

We can use the abovely defined GP as a prior over functions, simply to draw some candidate functions independently of having seen any data. Let us firstly denote by $x_\star$ a vector of n many inputs

$$x_\star = (x_1 \dots x_n) \tag{6.6}$$

of inputs where we are interested - denoted by the star - in observing the performance of these inputs. In most cases, we will only be interested in observing only one input, however, in order to draw functions from the prior we need many points to give the function enough support. Similar to $x_\star$, by $f_\star$ we denote the vector of true function values on these inputs. Then, by the definition of the Gaussian process we know that

$$f_\star \sim \mathcal{N}\big(\mu(x_\star), k(x_\star, x_\star)\big) \tag{6.7}$$

where $\mu(x_\star)$ is the mean function applied component wise to all inputs and the covariance matrix to be understood as pairwise evaluated over all pairs. As shown in [68], we can without loss of generality assume the stochastic process prior to be centered which means that the mean function equals zero everywhere

$$\mu(x_i) = 0 \quad \forall x_i \in \mathcal{X} \ . \tag{6.8}$$

This assumption shortens the notation but offers the same expressivity. We will see that when we condition the performance f of a new configuration on our observations, the predicted mean will very unlikely be zero, at least not if observations in the vicinity of the new observation exist. Sampling from such a prior is easy, we can simply sample some data points, compute the covariance on these data points and finally sample from a multivariate Gaussian with zero mean and covariance specified by the kernel that we have computed. In Figure 6.1, samples from GP priors with two different kernel functions are shown.

6.1.2 Kernel Functions

As Gaussian processes in general do not lose prediction complexity when their mean is set to zero, the kernel function determines the complexity as well as some other

assumptions on the function space as for example smoothness. Out of these reasons, choosing the right kernel function for the right problem is crucial, which is why we will introduce a few kernel choices in the upcoming paragraphs.

The observation is assumed to carry independent identically distributed noise, which is reflected in the kernel function by adding an additional term

$$k(x, x') + \chi(x_i = x_j)\sigma_y^2 \tag{6.9}$$

where χ is an indicator function that returns one if its input is a true statement and zero otherwise. In doing so, we effectively add the observation noise σ_y^2 only to the diagonal of the kernel and only on observed points. Doing this also circumvenes numerical problems when inverting the kernel matrix for inference when measurements occur twice or are very similar, as they would make the columns linearly dependent.

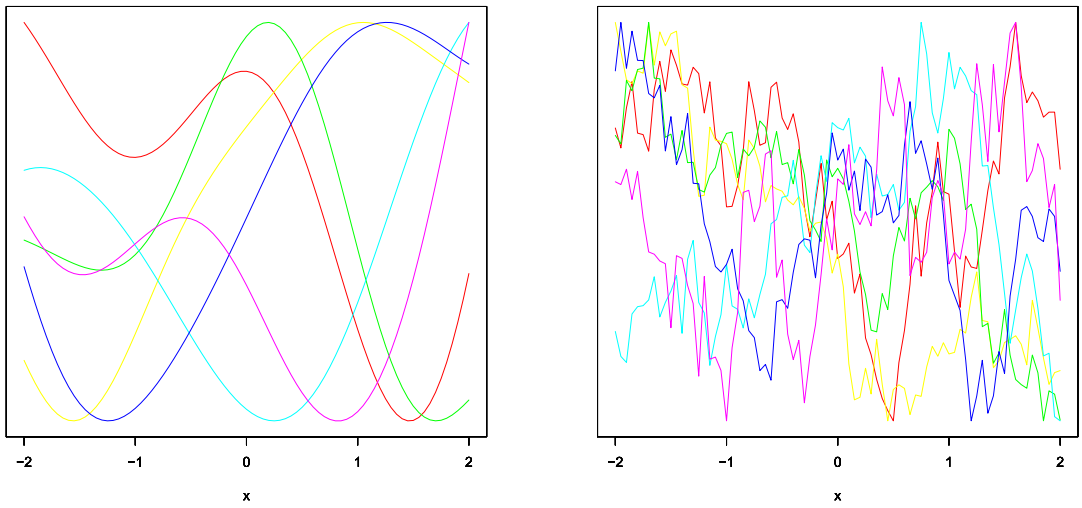

Figure 6.1: Comparison of candidate functions drawn from a Gaussian process prior. The left figure shows functions where the squared-exponential kernel has been used whereas the right figure employs the Dirichlet kernel. As can be seen, the squared exponential kernel delivers smooth functions while the Dirichlet kernel allows for sudden spikes and therefore non-smooth candidate functions.

A very common choice of kernel function is the squared exponentional (SE) kernel

$$k_{\text{SE}}(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{6.10}$$

which, based on the inputs computes their similarity. Inputs that are close to each other have a small distance $\|x - x'\|$, therefore the similarity grows as the negative squared

distance is inserted into the exponential function. As the mean function can be set to zero, the choice of the kernel function effectively defines the whole process and therefore the prior as well as the posterior. A squared exponential kernel for example enforces smoothness between measurements. For the current formulation of the SE kernel, all dimensions of x are weighted equally when computing the exponential term. However, for the similarity of two inputs x and x', some dimensions may have a higher influence, for instance one may see that the validation performance is varying much more in the setting of one hyperparameter than of the other. To model these differences in the similarity, the automatic relevance determination (ARD) kernel is used

$$k_{\text{ARD}}(x_i, x_j) = \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{1}{\sigma_i}(x_i - x'_i)^2\right) \quad . \tag{6.11}$$

The kernel hyperparameters σ_i are usually described as characteristic length scales as they measure the influence of each dimension individually. If for example $\sigma_i \longrightarrow \infty$, the dimension i does not influence the similarity anymore, as no matter how different x_i and x'_i are, the term effectively cancels out. Usually, using ARD correctly is not simple, as the number of hyperparameters - especially on data with lots of features - is simply too much. However, if the kernel hyperparameters σ_i can be estimated from the training data, using such a kernel may lead to better results.

There are more kernels that are similar to the SE kernel, for example the Dirichlet kernel

$$k_{\text{DIR}}(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|}{2\sigma^2}\right) \tag{6.12}$$

which looks very similar to the above kernel; however, it does not consider the squared distance, which does not lead to smooth functions anymore. Figure 6.1 shows a comparison of priors drawn from a Gaussian process with SE kernel and with Dirichlet kernel, where the measurement noise σ_f^2 has been set to zero. Clearly, choosing the SE kernel delivers smoother functions than the Dirichlet kernel.

Another kernel choice is the polynomial kernel of degree d constant term c, which is defined as

$$k_{\text{POL}}(x_i, x_j) = (x_i^\top x_j + c)^d \tag{6.13}$$

where, if we use a polynomial of degree $d = 1$, we obtain the linear kernel

$$k_{\text{LIN}}(x_i, x_j) = x_i^\top x_j + c \quad . \tag{6.14}$$

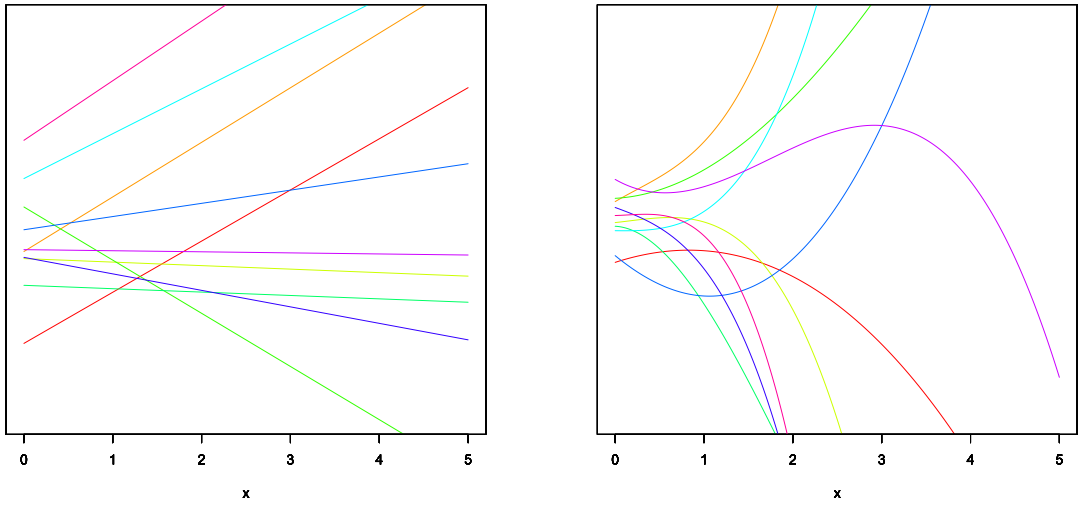

Figure 6.2: Comparison of candidate functions drawn from a Gaussian process prior. The left figure shows functions where the linear kernel has been used which results in only linear models being considered. The right figure uses the polynomial kernel of degree three, which is what we obtain as candidate models.

In Figure 6.2, samples from a GP prior with linear kernel and polynomial kernel of degree three are depicted. As one may suspect, we simply retain linear models from the linear kernel and polynomials of order d when employing the polynomial kernel.

6.1.3 Inference in Gaussian Processes

Now we are interested to infer the distribution of $f_\star$ for some unobserved input $x_\star$, given that we already have observed some data $y = (y_1, ..., y_n)$ for some inputs $x = (x_1, ..., x_n)$. This means we are interested in estimating the following conditional

$$p(f_\star \mid x_\star, x, y) \ . \tag{6.15}$$

As we know by the properties of the Gaussian process, the observed outputs y and the yet unobserved function value $f_\star$ are jointly Gaussian:

$$\begin{pmatrix} y \\ f_\star \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} K + \sigma_y^2 I & k_\star \\ k_\star & k_{\star\star} \end{pmatrix}\right) \tag{6.16}$$

where we define $K = k(x, x)$, $k_\star = k(x, x_\star)$ and $k_{\star\star} = k(x_\star, x_\star)$. Note that we have added the observation noise by adding a diagonal matrix to K. Using the results of Lemma

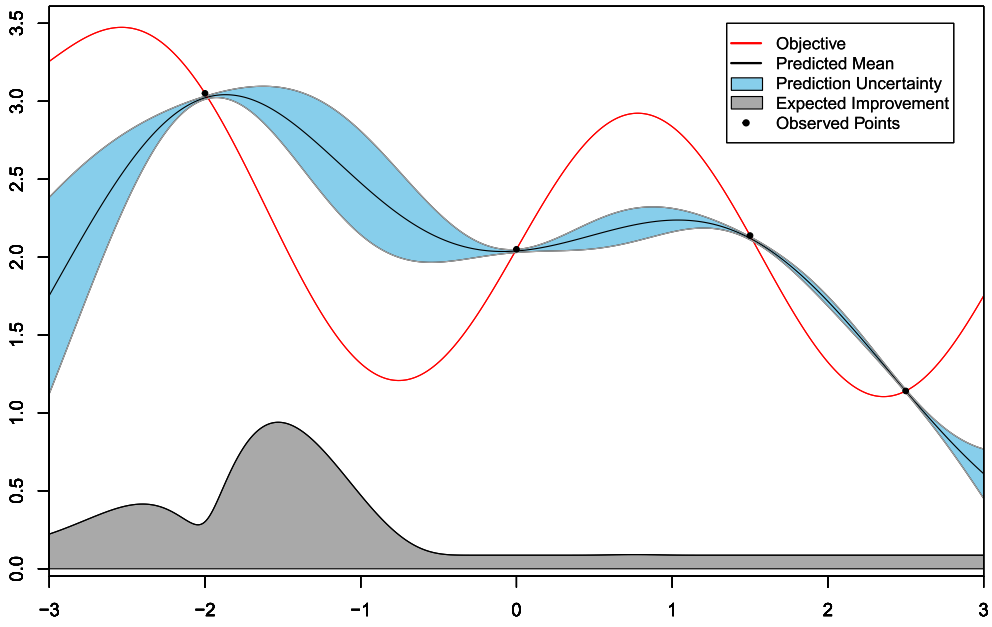

Figure 6.3: A Gaussian process prediction (black line) learned on four observations of a ground truth function (red line). The observations are reconstructed as there is almost no uncertainty (blue area). Far away from observed points the uncertainty increases. We can also see that on the boundary of the observed interval, the GP tends to have a zero mean.

A.3 from the appendix A on conditionals of multivariate Gaussians, we know that $p(f_\star \,|\, x_\star, x, y)$ is a Gaussian

$$p(f_\star \,|\, x_\star, x, y) = \mathcal{N}\big(\mu(f_\star), K(f_\star)\big) \tag{6.17}$$

with mean and covariance given by

$$\mu(f_\star) = k_\star^\top (K + \sigma_y^2 I)^{-1} y \tag{6.18}$$

$$K(f_\star) = k_{\star\star} - k_\star \top (K + \sigma_y^2 I)^{-1} k_\star \tag{6.19}$$

where we use the fact that we have assumed a GP prior with zero mean. An example of the conditional distribution of labels of a GP learned on a few data instances can be seen in Figure 6.3.

As we see from the mean and covariance formulas, an inversion of the quadratic kernel matrix K is necessary to compute the mean prediction and the uncertainty.

Unfortunately, the size of the kernel matrix equals the number of observed points which makes the Gaussian process computations infeasible for large data sets as the inversion operation is cubic in the size of the matrix to inverse. When running SMBO, we will increment our observation history $\mathcal{H}$ by one observation for each trial, which means that the inversion of K will consume more and more time throughout the SMBO optimization. In the beginning this might not be a problem but if the inversion takes almost the same amount of time as evaluating f simply because the meta knowledge is too large, this becomes infeasible.

6.1.4 Efficiently Updating the Kernel Inverse Sequentially

Now we will assume that we have learned a GP on some meta data meaning that we have computed the kernel matrix and its inverse. Then, we want to conduct SMBO optimization of hyperparameters with it, where in every step we will observe the hyperparameter performance of one additional configuration, and then will have to relearn the model. Naively, we could compute the kernel matrix again and invert it again, neglecting all information about it from the previous iteration. Here, we will present a way of estimating the new kernel inverse, which is only quadratic in the size of K and not cubic.

As K is a symmetric matrix, we can use the Cholesky decomposition of K to speed up computations.

Lemma 6.1. *Every symmetric, positive definite matrix A allows a decomposition*

$$A = LL^\top \tag{6.20}$$

where L is a lower triangle matrix, meaning the entrances $L_{ij} = 0$ if $j > i$. This decomposition is called Cholesky Decomposition [93].

Computation of the cholesky decomposition is cubic in the size of the matrix to be decomposed, but for our purposes, we only have to do the cubic operation once which is in the beginning. For the mean, we are interested in finding

$$\mu(f_\star) = k_\star^\top (K + \sigma_y^2 I)^{-1} y \tag{6.21}$$

so by setting $K + \sigma_y^2 I = LL^\top$ we can solve first

$$Lz = y \tag{6.22}$$

by forward substitution, and then solve

$$L^\top \alpha = z \tag{6.23}$$

by backward substitution. Then, we can simply compute the dot product of $k_\star$ and $\alpha = (K + \sigma_y^2 I)^{-1} y$ which gives the mean $\mu(f_\star) = k_\star^\top \alpha$. For the predictive variance, we can solve

$$Ll = k_\star \tag{6.24}$$

and then compute the resulting variance $K(f_\star) = k_{\star\star} - l^\top l$.

Now we want to update the kernel inverse after having observed a new instance, so effectively we want to update L. We can do so by setting

$$L_{\text{new}} = \begin{pmatrix} L & 0 \\ l & l_\star \end{pmatrix} \tag{6.25}$$

where we only have to compute $l_\star$ as

$$l_\star = \sqrt{k_{\star\star} - \|l\|_2^2 + \sigma_y^2} \quad . \tag{6.26}$$

In this way, we do not have to compute the Cholesky decomposition as many times as we want to conduct trials. We have to compute L only once in the very beginning, then we have to update it, and in every step solve the three systems of equations to allow us to make inference. Solving these systems of equations has an effort of $\mathcal{O}(n^2)$, which is a reduction by one order of magnitude of n.

6.1.5 Estimating Kernel Hyperparameters

A positive aspect of Gaussian processes is that they are essentially hyperparameter free, which is a great property if they are to be used as surrogates for hyperparameter optimization of another machine learning model. Naturally, using a model to predict hyperparameter performance of a target model only makes sense if the surrogate model uses no or less hyperparameters, or if its performance is less sensitive with respect to its hyperparameters, making them easier to optimize. Another possibility may be to simply learn them, which is the case for Gaussian processes as [68] shows. By κ let us define the kernel hyperparameters of a GP, so for the example of learning a polynomial kernel, the hyperparameter would be the degree of the kernel. For an ARD kernel, we have as many hyperparameters as features in the data.

We can efficiently optimize the hyperparameters κ by maximizing the log marginal likelihood on the training data. Given that $y \sim \mathcal{N}(0, K + \sigma_y^2 I)$, the marginal log likelihood is

$$\log p(y \mid x, \kappa) = \frac{1}{2} y^\top K_y^{-1} y - \frac{1}{2} \log |K_y + \sigma_y^2 I| - \frac{n}{2} \log 2\pi \tag{6.27}$$

where we have used $K_y = K + \sigma_y^2 I$ to shorten the notation. As we see, only the first and the second term actually depend on the kernel hyperparameters, the third term is obviously constant and will disappear when deriving the upper equation with respect to κ. The first term can be understood as a term that measures how well the data is fitted, whereas the second term involving the determinant of K_y resembles the complexity of the model. Computing the hyperparameter gradients of the marginal likelihood with respect to κ_j yields

$$\begin{aligned} \frac{\partial}{\partial \kappa_j} \log p(f \mid x, \kappa) &= \frac{1}{2} \frac{\partial}{\partial \kappa_j} K_y^{-1} - \frac{1}{2} \frac{\partial}{\partial \kappa_j} |K_y| \\ &= \frac{1}{2} y^\top K_y^{-1} \frac{\partial K_y}{\partial \kappa_j} K_y^{-1} y - \frac{1}{2} \operatorname{tr}\left(K_y^{-1} \frac{\partial K_y}{\partial \kappa_j} \right) \\ &= \frac{1}{2} \operatorname{tr}\left((\alpha \alpha^\top - K_y^{-1}) \frac{\partial K_y}{\partial \kappa_j} \right) \end{aligned} \tag{6.28}$$

where we again use the notation $\alpha = K_y^{-1} y$, and used the rule of the derivative of the inverse matrix as well as deriving the log determinant

$$\frac{\partial}{\partial \theta} A^{-1} = -A^{-1} \frac{\partial A}{\partial \theta} A \qquad \frac{\partial}{\partial \theta} \log |A| = \operatorname{tr}\left(A^{-1} \frac{\partial}{\partial \theta} A \right) \ . \tag{6.29}$$

Given a specific kernel choice, we only have to compute its derivative with respect to the kernel hyperparameters and then we are able to update all parameters. As we see, the computation that it the most time consuming is the inversion of the kernel matrix, but once this has been done, it does not matter too much how many kernel hyperparameters we want to optimize. Therefore, using kernels such as ARD does make sense in our scenario.

6.2 Product of Experts Models for Hyperparameter Optimization

In this section, we will have a look at different product of experts models, where we first have a look at the initial formulation of products of experts. Then, we will look at generalized products of experts which basically only include an additional weighting term to each of the experts. Afterwards, we will have a short look at Bayesian committee machines, which try to estimate a slightly different distribution.

6.2.1 (Generalized) Product of Experts

From now on, we will assume that we have partitioned our meta data set into M many pairwise disjoint parts

$$X = (X^1, ..., X^M) \qquad y = (y^1, ..., y^M) \tag{6.30}$$

where again, the output y is a noisy measurement of a true underlying function f. We will also use the notation

$$D^i = (X^i, y^i) \tag{6.31}$$

when we are specifically talking about the training data for the i-th expert.

There are many ways to split data this way, we will use the most natural and most obvious split in our application. As our meta data consists of observations of hyperparameter performance on many different problems (data sets), we can simply divide the data problem-wise.

A product of experts model as proposed in [35] factorizes a probability density $p(y|X,\theta)$ parametrized by θ over a complete data set (X,y) as

$$p(f|X,\theta) = \prod_{i=1}^{M} p(y^i|X^i,\theta_i) \tag{6.32}$$

a product of individually learned densities. Now, each of these individual densities can be modeled by a Gaussian process, which then only has to be learned on a partition of the data and not the whole data. This will make the inversion of the kernel matrix much faster, as the kernel will only be evaluated on a subset of data points. The way we have partitioned our data problem-wise also seems natural, as now we will learn a model for the response surface of each problem individually, making this model an expert on that very problem. By combining all of these experts in the end, we can generate an ensemble of experts who then decide what to do next.

The generalized product of experts [10] introduces additional weighting factors β_i as

$$p(f|X,\theta) = \prod_{i=1}^{M} p^{\beta_i}(y^i|X^i,\theta_i) \tag{6.33}$$

where if we set all $\beta_i = 1$, we retain the initial formulation of the product of experts. There are some methods on estimating decent values for β_i, for example based on entropy differences of the prior and posterior, as is done in [10]. However, if all experts are assumed to be equally strong (or equally weak), a setting of $\beta_i = 1/M \; \forall i$ works

suprisingly well. In our scenario, we currently have M response surfaces of equal size, additionally all GPs are learned by optimizing their kernel hyperparameters, so it is safe to assume that all experts perform more or less equally well.

If we assume a Gaussian process for each individual conditional, by combining all of them we will have to multiply M many Gaussians to obtain the final density. As we have already seen in Chapter 3, the product of abritrary many Gaussians is again a Gaussian with mean and variance

$$\begin{aligned} \mu^{\text{gpoe}}(f_\star) &= (\sigma^{\text{gpoe}}(f_\star))^2 \left(\sum_{i=1}^{M} \beta_i \sigma_i^{-2}(f_\star) \mu_i(f_\star) \right) \\ (\sigma^{\text{gpoe}}(f_\star))^2 &= \frac{1}{\sum_{i=1}^{M} \beta_i \sigma_i^{-2}(f_\star)} \quad . \end{aligned} \tag{6.34}$$

By inspecting the mean one more time, we see that it actually is a weighted sum of individual means

$$\mu^{\text{gpoe}}(f_\star) = \frac{\sum_{i=1}^{M} \beta_i \sigma_i^{-2}(f_\star) \mu_i(f_\star)}{\sum_{i=1}^{M} \beta_i \sigma_i^{-2}(f_\star)} = \frac{\sum_{i=1}^{M} w_i \mu_i(f_\star)}{\sum_{i=1}^{M} w_i} \tag{6.35}$$

where each weight is defined by the coefficient β_i and the precision of the i-th model

$$w_i = \beta_i \sigma_i^{-2}(f_\star) = \frac{\sigma_i^{-2}(f_\star)}{M} \tag{6.36}$$

For the mean, it actually does not make a difference if we set β_i to one or to the reciprocal of the number of experts, as both terms effectively cancel out. However, the variance is different, as the division by M only makes the denominator smaller, and therefore increases the whole expression. According to [17], this improves the model overall, as in many cases it is overconfident in areas where it has not seen much data and therefore likely does not explore well.

6.2.2 (Robust) Bayesian Committee Machines

A slight variant to product of experts models was given by Tresp in [91], where Bayesian committee machines (BCM) are introduced. The derivation is slightly different, as Tresp assumes the data for the i-th and the j-th expert to be independent, given the response $f_\star$. As Tresp outlines, this assumption is valid if the data sets D^i and D^j are spatially separated from each other. In our case this is not necessarily true, as hyperparameters are still the same for each problem, however, by including meta features we can spatially

separate them. The derivation of the Bayesian committee machine for only two experts is as follows

$$\begin{aligned} p(f_\star | D^i, D^j) &\propto p(D_i, D_j | f_\star) p(f_\star) \\ &= p(D_i | f_\star) p(D_j | f_\star) p(f_\star) \\ &= \frac{p(D_i, f_\star) p(D_j, f_\star) p(f_\star)}{p(f_\star) p(f_\star)} \\ &\propto \frac{p(f_\star | D_i) p(f_\star | D_j)}{p(f_\star)} \end{aligned} \tag{6.37}$$

where we use the independence assumption in the second equation. If we would look at the derivation for three experts, we will have the same transformations, but will end up having the prior probability of $f_\star$ squared. In general, for M many experts the model has the form

$$p(f_\star | D^1, ..., D^M) \propto \frac{\prod_{i=1}^M p(f_\star | D_i)}{p(f_\star)^{M-1}} \quad . \tag{6.38}$$

In [17], Deisenroth and Ng extend the BCM to the robust BCM, this is done by simply adding coefficients β_i to the model, as was also done for the generalized product of experts. Adding these terms yields a final model

$$p(f_\star | D^1, ..., D^M) \propto \frac{\prod_{i=1}^M p^{\beta_i}(f_\star | D_i)}{p(f_\star)^{(1-\sum_{i=1}^M \beta_i)}} \tag{6.39}$$

where, if we set all $\beta_i = 1$, we retrieve the initial formulation of BCMs. As we have seen already for the POE models, if we multiply all these Gaussians, we obtain a density that is proportional to a Gaussian with mean and variance

$$\begin{aligned} \mu^{\text{rbcm}}(f_\star) &= (\sigma^{\text{rbcm}}(f_\star))^2 \left(\sum_{i=1}^M \beta_i \sigma_i^{-2}(f_\star) \mu_i(f_\star) \right) \\ (\sigma^{\text{rbcm}}(f_\star))^2 &= \left(\sum_{i=1}^M \beta_i \sigma_i^{-2}(f_\star) + (1 - \sum_{i=1}^M \beta_i) \sigma_f^{-2} \right)^{-1} \end{aligned} \tag{6.40}$$

where σ_f^{-2} is the prior precision, which is simply the reciprocal of the kernel evaluated on the new point $k_{\star\star}$. For a Gaussian kernel, this equals one, as the metric in the exponential becomes zero, meaning that both variance and precision are the same.

Interestingly, when we are setting $\beta_i = 1/M$, we get the same model as the generalized product of experts, as the additional prior variance will simply cancel out. Therefore we

have to set β_i to different values, we follow the proposition by [17], who use

$$\beta_i = \frac{1}{2}\left(\log\sigma_f^2 - \log\sigma_i^2(f_\star)\right) = \frac{1}{2}\left(\log(1+\sigma_y^2) - \log\sigma_i^2(f_\star)\right) \tag{6.41}$$

where the second equality only holds for kernels that have a similarity of one for equal data points. We introduce both BCM models for the sake of completion; however, we did not find them to be applicable for our problem while running the experiments, as for the first SMBO runs the estimated precisions turned out to be negative. This is due to the fact that we use a reasonably large set of meta features and therefore the distance between known and unknown hyperparameter observations is quite large, leading to larger uncertainties at the beginning of SMBO. A standard POE model does not get harmed by this, however, as BCMs subtract the prior precision from the ensembles precisions as in equation 6.40, this effect occurs and delivers non-interpretable results. This effect can be prevented by using less meta features, which will lead to worse results as the surrogate models have less information to distinguish between data sets, which is an effect that we would like to avoid. Due to this fact, BCMs will be left out in our experiments.

6.2.3 POE for Hyperparameter Optimization

So far, we have discussed several product of experts models, as well as ways to learn their kernel hyperparameters and ways to compute conditionals of the performance of yet unobserved points. Thus, applying POE to hyperparameter optimization seems straightforward and easy. However, there are some additional questions, that have to be answered.

At first, the way how we split the data needs to be discussed. We can simply split it randomly, however, this partition does not seem that natural. In order to really learn experts on some part of the data, we think it makes most sense to distribute the data problem-wise. Say, we know the response surface of an SVM on five data sets, why should we not directly split on the data sets. There may only be the scenario, where even the meta knowledge of one data set is already too much, in our experiments we have not yet faced this, but if we ever end up in this scenario, we can simply divide all the meta knowledge of one problem into two sets and then simply learn two experts.

Another question that specifically shows up in SMBO is how we make use of the additional data that we observe after each new trial. This is crucial, as the new data points are the ones that have the highest influence and where we are mostly interested in. For the experiments in this section, we have designed two alternatives. In the first

alternative, we simply relearn all M experts by giving every expert the newly observed point, in this way every expert becomes an expert for his data plus for the target data. We will simply call this method POGPE, as it is a product of GP experts.

As another alternative, we consider learning an $M+1$-th expert, that only sees the new data while withholding the new data from the other experts. In order for the target GP to not get outvoted by all other experts, we then set the weight to $\beta_i = 1/2M$ for all experts on the already observed meta data, and $\beta_{M+1} = 1/2$. The correct setting of these weights is of course a surrogate hyperparameter and maybe having only 50% of the overall vote is not enough for the single expert. We call this method SGPE as abbreviation for single Gaussian process expert, although the name is a little ambiguous as the decision is still made by an ensemble of experts.

One final consideration that is crucial is to scale the response surface to unit variance. If different response surfaces are highly different in their scale, the POE does not work. This may occur, if one problem is very simple to solve, where even the hyperparameter has almost no influence and almost all configurations get a high accuracy, opposed to a problem where only few hyperparameters work well. As we cannot scale the hyperparameter response on the target data, we do this by computing empirical means and standard deviations on the fly and using these values to scale the response values. This was also proposed by Yogatama and Mann in [96].

6.2.4 Empirical Evaluation of POE-based Hyperparameter Optimization

Now we will present the results of the paper *Scalable Hyperparameter Optimization with Products of Gaussian Process Experts* [80] published at ECML. As we have already mentioned, this chapter does not end afterwards, as it also introduces the additional work done by Martin Wistuba and me, which resulted in a journal submission [95].

Performance in SMBO At first, we will have a look at the performance in SMBO, which is the main task we want to conduct with our proposed surrogate models. For a detailed review of all competing surrogate models, we refer the reader to Chapter 3.

Figure 6.4 shows the average rank among all competing methods, on the left side, the result for AdaBoost is shown, on the right side the results on the SVM meta data. As we have already seen in other experiments, the random surrogate has the worst performance, so this is not unexpected at all. POGPE and SGPE both work surprisingly

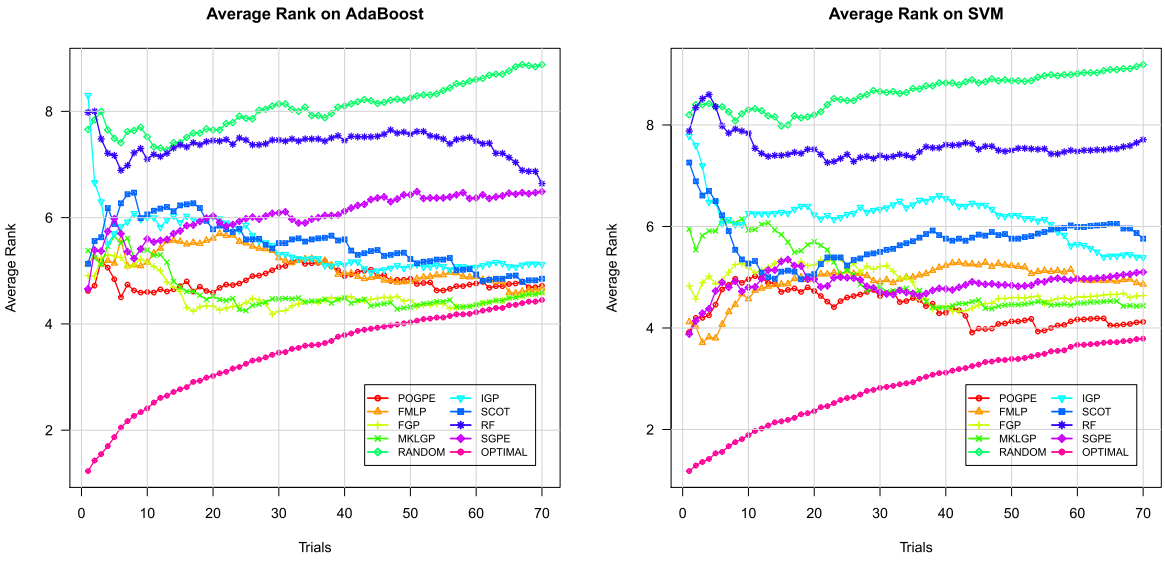

Figure 6.4: Average Rank of all competing methods. The left plot shows results for AdaBoost, the right plot shows results for the SVM meta data set.

well in the beginning of SMBO, as both methods outperform all other methods in the very first trial. SGPE, however does not seem to work that well, as besides its very good start, it starts to deteriorate and gets beaten by the full GP as well as MKLGP. For POGPE, this is also the case on AdaBoost, however, on SVM it suprisingly still works better than its GP competitors MKLGP and FGP, where we also have to acknowledge that FMLP works best on SVM in the first 15 trials. FMLP however degrades and performs worse than all the GP-based methods, except for the independent GP of course. Interestingly, both FGP and MKLGP have a very similar performance, meaning that the additional data set kernel used in MKLGP does not have too much of an influence, which may be due to the choice of metafeatures.

In contrast to POGPE, SGPE does not show a very good performance in general. It has a very nice start, due to it being the same as POGPE in the very first trial, but then the performance deteriorates which is likely due to how the single expert is weighted into the ensemble. Setting its vote to 50% of the overall votes may be too much in the beginning, as having seen a few data points does not convey too much knowledge of the response surface. Conversely, going towards the later trials, the weight may be too low, and the single expert is only being confused by the other experts. Finding a suitable weighting scheme for SGPE is one of the contributions that will be shown later in this chapter.

We conclude that FGP, MKLGP, FMLP and POGPE are the best working surrogates

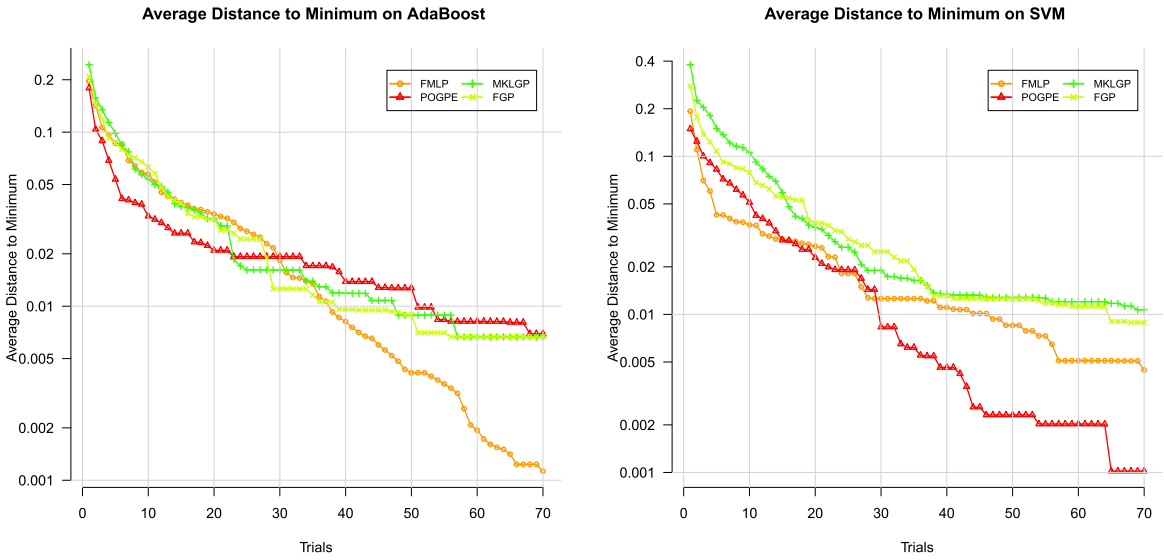

Figure 6.5:]
Average Distance to Minimum of the best performing methods. The left plot shows results for AdaBoost, the right plot shows results for the SVM meta data set.

presented so far, therefore we have also evaluated the average distance to the minimum for these methods. The average distance to the minimum computes the distance in performance values between the best working hyperparameter configuration found so far and the overall optimum of the grid. These values are then averaged over all data sets and random runs. The results are shown in Figure 6.5, where again AdaBoost is shown on the left and SVM on the right. What we can see immediately is that optimizing hyperparameters for AdaBoost is simpler as the average distance to the minimum of all methods is around 0.2, wheras this value is almost doubled for SVM. Please note that the y-axis is in log scale to show also the tiny improvements achieved in the later trials. FMLP seems to do very well after trial 40 on AdaBoost, however, this performance improvement is due to it winning severly on the data set *sonar-scale* against the other surrogates. This is still being reflected in the average distance to the minimum (especially on a log scale) but gets concealed in the average rank, which is why FMLP does not work that well in Figure 6.4. POGPE seems to work best on SVM, however, this time it is not due to solving a specific problem, as the good performance is also refelected in the average rank. Overall, we conclude that POGPE works really well, especially when considering its simplicity, its easy way to be parallelized and the fact that is actually only an approximation of a full Gaussian process.

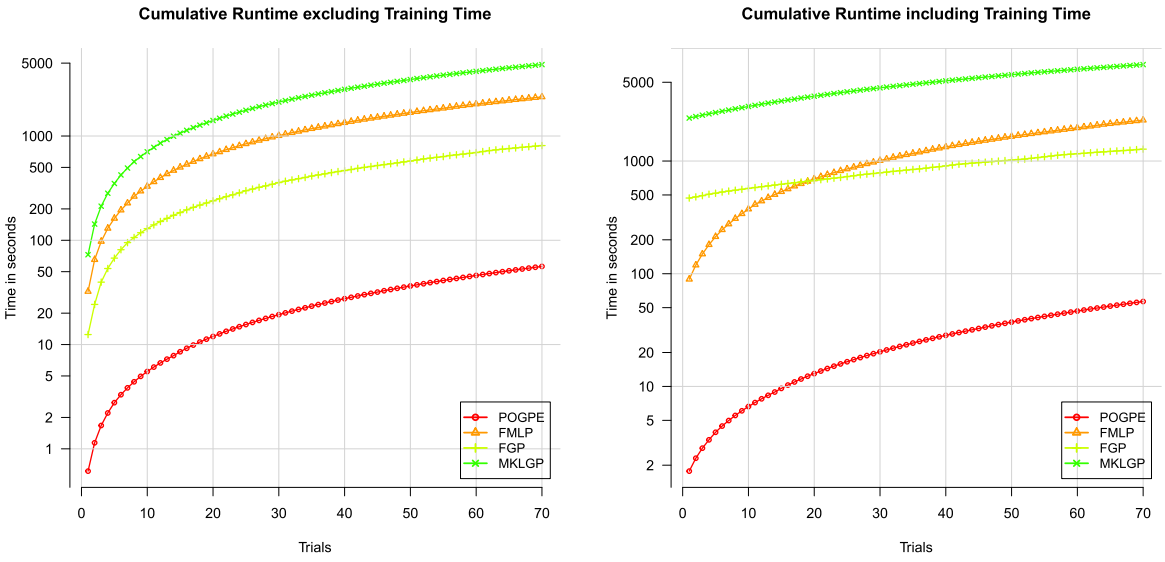

Figure 6.6: Runtime comparison among the most competitive surrogate models. The left plot shows the cumulative runtime in seconds.

Runtime Comparison In order to determine the scalability properties of our proposed surrogate model, we have measured the runtime of the four best performing methods, which are POGPE, FGP, FMLP and MKLGP. The experiments were conducted on a Xeon E5-2670v2 with 2.5GHz clock speed and 64GB RAM. Similar to the performance experiments, we again conduct 70 trials of SMBO on the SVM meta data and average the runtimes over ten runs to account for measurement noise.

The results can be observed in Figure 6.6, where we have measured the runtime for two scenarios. Please note that the y-axis is again in log scale, meaning that the actual differences are much larger. On the left, we show the cumulative runtime in seconds while excluding the time it took to learn the surrogate model on the meta data, consequently, the right plot shows the cumulative runtime including training time. We show both scenarios, as one can imagine learning the surrogate model already in an offline fashion where one does not care too much about the runtime and only cares for how long the surrogate update takes. For both plots, we do not include the runtime of evaluating f, which is problem dependent anyway. Therefore, the runtime can be seen as total overhead time one would use for SMBO iterations over doing something else like a grid-search for example.

What immediately jumps in the eye is the fact that POGPE uses drastically less time than all other competitors. For POGPE, the overhead time is only around 35 seconds which is amazing compared to the other methods where the overhead is around ten

minutes for the FGP up to more than one hour for MKLGP and FMLP being somewhere between those values with around half an hour of time used. If we also include the time for training the surrogate model, the full GP takes more time in the beginning, simply as learning the surrogate on the meta data is quite time consuming. In this scenario, using FMLP saves some time, but this advantage is lost after trial 20. For POGPE, including training time or not does not make much of a difference, as learning the surrogate on the meta data only takes around one second. We conclude that POGPE scales very well, especially in the scenario envisioned in the beginning of this chapter. With more and more meta data being created every day, we are in need of scalable but powerful surrogate models.

6.3 Transfer Surrogate and Acquisition Framework

This section introduces the additional work that has been done on POE models in the journal submission *Scalable Gaussian Process-based Transfer Surrogates for Hyperparameter Optimization* in the machine learning journal.

6.3.1 Transfer Surrogate Models

Now we want to generalize over the product of experts models that have been introduced already in this chapter, first by defining the model class in a more general way, and then by introducing different ways of estimating weighting coefficients. We again assume that we can split our observations into M many parts such that all parts are pairwise disjoint, i.e. every observation is in exactly one part X^i

$$X = (X^1, ..., X^M) \qquad y = (y^1, ..., y^M) \ . \tag{6.42}$$

We call a surrogate model a *scalable Gaussian process transfer surrogate* (SGPT), if its mean prediction and precision has a form

$$\begin{aligned} \mu(f_\star) &= \frac{\sum_{i=1}^{M+1} w_i \mu_i(f_\star)}{\sum_{i=1}^{M+1} w_i} \\ \sigma^{-2}(f_\star) &= \sum_{i=1}^{M+1} v_i \sigma_i^{-2}(f_\star) \end{aligned} \tag{6.43}$$

and if there is an additional indicator $v \in \{0,1\}^{M+1}$ that defines which expert has access to the observations on the target response surface, i.e. which expert is being fitted to the

targets observed throughout the optimization procedure. Please note that we drop the dependency of the coefficients w and v on $f_\star$. We will see that some of our weights are actually dependent on $f_\star$, we only drop the additional dependence to avoid unnecessary clutter.

Similar to the single Gaussian process expert, we now assume learning $M+1$ experts, where the first M experts are learned on the meta data. For the observations on the target response surface we define an indicator variable ν that indicates, whether the i-th expert has access to the observations on the target response surface By setting ν as

$$\nu_i = 1 \quad \forall\, i = 1,...,M \quad \nu_{M+1} = 0 \tag{6.44}$$

we retain the initial product of experts, and conversely by setting it to

$$\nu_i = 0 \quad \forall\, i = 1,...,M \quad \nu_{M+1} = 1 \tag{6.45}$$

we obtain the single Gaussian process expert, both models were explained in the previous section. From now on, we will assume that we use the former setting of ν, meaning that only the target GP gets to see the new data. In the experiments in the previous section, the SGPE has shown to perform not so well, however, we argue that this is due to the poor setting of the ensemble weights. In the following, we will derive different ways of setting these ensemble weights in a more optimal manner than before.

Product of Experts

As already mentioned, it is simple to retain the PoE models that have been used in [80] by defining ν, w and v in the correct ways. For the experiments to follow, we test a different parametrization, where we set

$$\begin{aligned} w_i &= \frac{1}{M+1}\sigma_i^{-2}(f_\star) \\ v_i &= \frac{1}{M+1} \quad . \end{aligned} \tag{6.46}$$

One major issue of setting the ensemble weights as above is the fact that they are static and do not change througout the optimization procedure. Another problem is that the vote of each experts is weighted equally which in reality does not make too much sense. Considering that the geometry of one response surface can be more closely related to the target response surface than some other response surface, we would naturally want to upweight the former expert in comparison to the latter one. We see how this can be done by introducing another SGPT model.

Kernel Regression

Now we will shortly introduce the work by Martin Wistuba in [94], which fits exactly into the abovely proposed SGPT framework. As said before, we now want to derive ensemble weights that give higher weight to data sets which have a similar hyperparameter response surface compared to the target's response surface. Naturally, one way of doing so is by defining data set descriptors, i.e. metafeatures on the data sets. Assuming we have a set of data set descriptors m_i, m_{M+1} for data set D_i and the target data set D_{M+1} respectively, then we could simply compute the ensemble weight w_i using a kernel function

$$w_i = k(m_i, m_{M+1}) \ . \tag{6.47}$$

The kernel used in [94] is the Epanechnikov quadratic kernel [22]

$$\begin{aligned} w_i = k(m_i, m_{M+1}) &= \gamma\left(\frac{\|m_i - m_{M+1}\|_2}{\rho}\right) \\ \gamma(t) &= \begin{cases} \frac{3}{4}(1-t^2) & \text{if} \quad t \leq 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{6.48}$$

if we set our weights like this, the scalable surrogate transfer model becomes equal to a kernel regression with the Nadaraya Watson kernel-weighted average, at least for predicting the mean. Now, computing the weights v of the precision could be done in a same manner, however, [94] propose to trust only the expert learned on the target response surface, i.e. setting v as

$$v_i = 0 \quad \forall i = 1, ..., M \qquad v_{M+1} = 1 \tag{6.49}$$

which seems to work well.

The authors propose two ways of estimating data set similarities, once by setting m_i to the meta features computed from data set D_i. Secondly, similarities are being computed in an adaptive fashion by a pairwise ranking approach, that is based on the t many hyperparameter configurations $x_1, ..., x_t$ that have been observed already on the target data set. The intuition is to compute the similarity by counting how many times the ranking of these configurations are equal. As there will be scenarios where we test new configurations that have not been observed for all previous problems, we simply take the mean prediction of the respective expert to compute the ranking. More formally, given a data set D_i, its meta features $m_i(t)$ at trial t are computed, by considering all

$t(t-1)$ pairs (x_k, x_j) and defining the entries of $m_i(t)$ as

$$(m_i(t))_{j+(k-1)} = \begin{cases} \frac{1}{t(t-1)} & \text{if} \quad \mu(f_k) > \mu(f_j) \\ 0 & \text{otherwise} \ . \end{cases} \tag{6.50}$$

Please note that we consider all $t(t-1)$ pairs, so we will take the pair (x_1, x_2) into account as well as the pair (x_2, x_1). Computing the euclidean distance as in the Epanechnikov kernel, we obtain the number of discordant pairs, this measure is equal to the ranking coefficient by Kendall [46].

The big advantage of the former way of defining meta features based on the response surfaces of each problem is twofold. First, it expresses similarity directly on the values we are actually interested in, and not by defining some peculiar meta features, where the dependence to the response surface is rather limited. Secondly, by having adaptive meta features, the weighting is not carved in stone but adjusts itself to the performances observed. We will see very soon, that both of these advantages boost the resulting performance by quite some margin.

6.3.2 Transfer Acquisition Functions

In the following pages, we will introduce the *transfer acquisition function framework*, which builds on the aforementioned SGPT framework, but improves on two drawbacks. At first - and this is a problem of very many surrogate models not only proposed in this thesis but also in the literature - by learning either a joint surrogate or a product of experts, we always try to reconstruct the individual response surfaces. This leads to the problem that our surrogate models learn on different scales, and therefore an expert of a hard data set might have a small influence when solving a rather simple data set, simply because this experts expects much less out of testing different configurations.

A common workaround for this problem is to simply scale the data, which has two inherent problems. At first, since we are completely unaware of the target response surface, doing something like on-the-fly scaling with a couple of data points is usually highly inaccurate and may lead to poor decision making in the beginning of the SMBO runs. This is always a problem as the initial performance has a huge effect on the final configuration we end up with. Secondly, we assumed initially that the meta data is highly heterogeneous, meaning that the hyperparameter grids employed for earlier problems can differ a lot, which introduces a bias into the scaling as well, since we can never be sure to have found the overall amplitude of the response surface, as well as its most interesting regions. The second problem, however, is only partly relevant, as for our

experiments, we have used the same grid over all past problems we solved. In a real world application this will likely not be the case.

Another issue with the proposed SGPT framework is that although it already considers adaptive weights of each individual expert, the influence of each expert is still being modeled explicitly. In many experiments in this thesis, we have seen a rather strong performance by the independent GP surrogate, which only learns a GP on the target observations. This very simple model actually performs well after it has seen some observations on the target data set, it is usually then already outperformed by surrogates using meta knowledge, which in some scenarios is only helpful at the beginning. Therefore we propose a framework where the influence of meta data can actually become zero.

The *transfer acquisition function framework* (TAF) suggests to learn a product of experts as introduced above, however, the voting phase is not before evaluating the acquisition function, but within evaluating it. We therefore propose to compute the acquisition function as

$$a(x) = \frac{\sum_{k=1}^{M+1} w_k a_k(x)}{\sum_{k=1}^{M+1} w_k} \tag{6.51}$$

where a_k is the acquisition function for the k-th expert. Naturally, we will assume that all acquisition functions are equal, in our upcoming experiments we will use the expected improvement as acquisition function. We can set the weights in analogy to the POE and kernel regression models proposed above.

While the framework looks very similar to the aforementioned SGPT framework, it behaves differently because of two effects that to a certain degree benefit from each other. At first, since we are not directly modelling the hyperparameter response, add it to some mean prediction, and then ask for the acquisition function, we ask each expert for the improvement beforehand. This leads to very many scenarios where the improvement of one expert is actually zero and therefore is not considered; however, the improvement will likely only be zero after some observations of the target problem have been made. This is exactly what we wanted to model, while in SGPT the influence goes to zero, it actually never becomes zero, and the convergence towards zero is only attained by choosing the right ensembling weights. For TAF, this problem is gone, as no matter what the coefficient is, if the EI is zero, the whole influence of the expert becomes zero.

Secondly, the uncertainty on the new problem is the same as for an independent GP, which is usually quite high in the beginning, leading to heavy exploration in the presence of no meta knowledge. For TAF, this effect is canceled out, in the beginning the

uncertainty of the target expert may be high but is balanced with the uncertainties of the meta experts. After a few iterations, the meta influence decreases rapidly, but at this point the uncertainty of the target expert is becoming more and more accurate, leading to well balanced decisions regarding exploration and exploitation. In this sense, we seek to combine the power of a single meta-independent surrogate model for the target data in the later stages with the meta knowledge being exploited in the earlier iterations.

6.4 Empirical Evaluation

In this section, we will empirically evaluate the SGPT as well as the TAF framework which we introduced earlier. Our experiments will go in three different directions, at first we will compare the performance in SMBO, which is the main purpose of our surrogate models. Secondly, we perform statistical tests on the observed performance, to assess how significant our surrogate models perform under test conditions. Finally, we show that SGPT and TAF, similar to the POE model proposed in [80] show very decent scalability properties compared to the remaining state of the art in Bayesian hyperparameter optimization.

6.4.1 Performance in Sequential Model-Based Optimization

First of all, we want to compare our firstly proposed transfer surrogate model (SGPT) within its model class, meaning we want to compare all of our proposed variants of SGPT between each other. Afterwards, we will have a look at the comparison to the current state of the art in order to see how SGPT performs overall. We have conducted two main experiments, one on the SVM meta data set which we have already seen in earlier experiments, another on the Weka meta data where we have not yet seen results in this thesis. We have left out the AdaBoost meta data set, as in our eyes it is simply too easy to solve, resulting in every surrogate model showing competitive performance, which lessens the amount of comparibility between individual tuning strategies. We show results of running 70 trials on the SVM meta data and for running 300 trials on Weka, where combined model choice and hyperparameter optimization are being conducted leading to a more difficult problem.

The average rank, average distance to the minimum as well as the fraction of unsolved data sets for both the SVM and the Weka meta data for all SGPT variants are shown in Figure 6.7 and 6.8. SGPT-PoE is the surrogate based on products of experts, SGPT-M

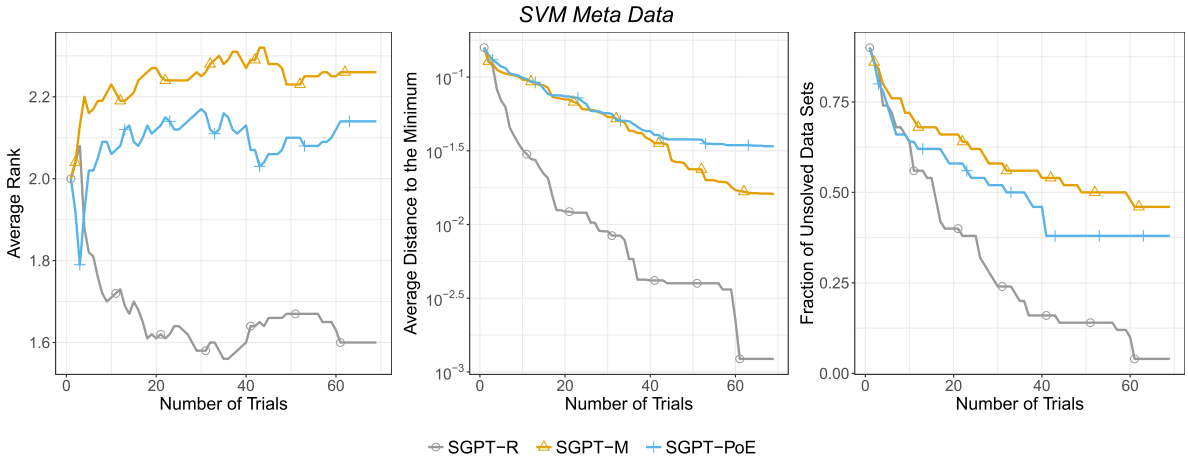

Figure 6.7: SGPT-R is outperforming the other two approaches due to the decaying influence of the meta-data for the task of hyperparameter optimization.

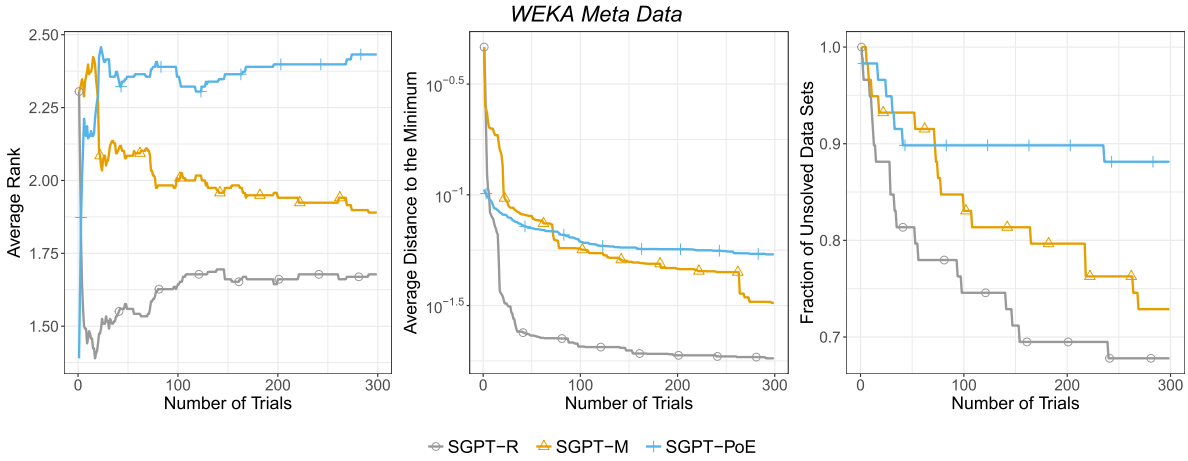

Figure 6.8: The adaptive weights also allows SGPT-R to outperform its variants for the task of combined algorithm selection and hyperparameter optimization.

describes the kernel regression using meta features and finally SGPT-R denotes the surrogate that estimates ensembling weights based on the pairwise ranking coefficient on the already observed response surface. As we can see, SGPT-R easily outperforms the other two surrogates in all of the three evaluation metrics on both meta data sets. SGPT-PoE arguably reaches second place on SVM, where it only loses to SGPT-M on the average distance to the minimum metric, which may be due to performing better on one single data set. On the Weka data set, SGPT-PoE is cleary worse than SGPT-M. However, both SGPT-PoE and SGPT-M gain second and third place with a high distance to the first place which is SGPT-R. We argue that the performance improvement shown by SGPT-R stems from the adaptively learned ensembling weights, which take similarities

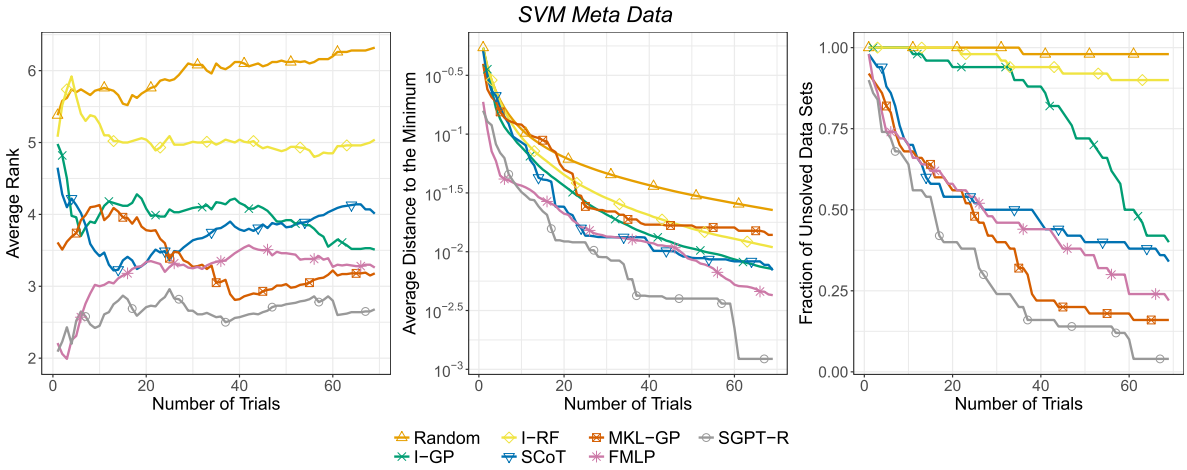

Figure 6.9: Our proposed approach SGPT-R is outperforming all competitor methods with respect to all three metrics. For all metrics, the lower the better.

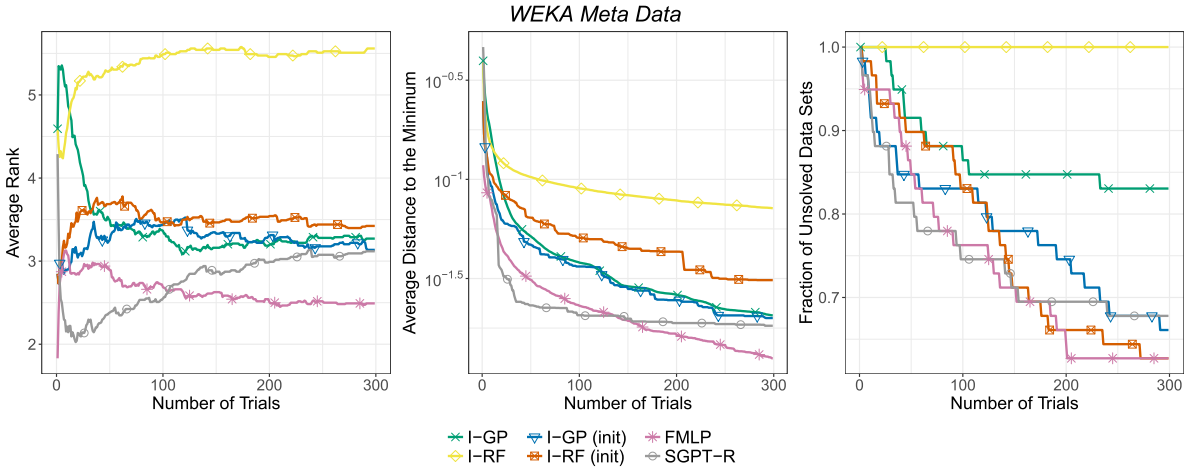

Figure 6.10: SGPT-R is outperforming the competitor methods for the task of combined algorithm selection and hyperparameter optimization only in the first iterations. Then, FMLP provides the best results. On this larger meta-data set we were not able to compare to those methods that are based on a GP that is trained on the whole meta-data (i.e. SCoT and MKL-GP).

of response surfaces, not of data sets into account.

Now, we will compare the best surrogate of the SGPT family, namely SGPT-R with the rest of the state of the art. We have left out both SGPT-M as well as SGPT-PoE as they have performed rather poorly in the previous experiment. Figure 6.9 and 6.10 show the performance against state of the art models used in hyperparameter optimization. Again, we show the average rank, average distance to the minimum and the fraction of unsolved data sets.

For the SVM meta data, we see a clear winner among all competing models which is

SGPT-R as it outperforms all other methods, except for FMLP in the very early SMBO iterations. This may be due to the fact that learning latent representations of the data set *given* the response surface is still a very good idea, which as future work could somehow be merged with learning scalable surrogate models. For SVM, it is also interesting to have a look at the independent GP as its behaviour in the later trials is what we wanted to encode in our surrogates SGPT-R and TAF. We see that after roughly 30 trials, I-GP is starting to perform much better, we can see this especially in the fraction of unsolved data sets.

For Weka, the picture is not entirely the same. SGPT-R seems to perform really well, but its closest competitor - FMLP - has caught up with regards to the experiments on the SVM meta data. This is likely due to the fact of the Weka data involving more categorical variables, which can be used more efficiently by FMLP. However, it needs some time for FMLP to improve, as it only catches up after around 150 trials, which is already half of the number of grid points we allow for the Weka experiment. Besides FMLP, SGPT-R is quite competitive, additionally given the fact that it easily scales to larger data sets, which we will see in the upcoming pages.

Now, we finally want to assess the performance of our transfer acquisition framework TAF. First, we compare all variants of TAF to all variants of SGPT in order to evaluate which framework and which setting of ensembling weights is more fruitful. The results on both meta data sets are shown in Figure 6.11 and 6.12, where we see that for the SVM meta data, both TAF and SGPT-R are quite competitive. We acknowledge however that TAF seems to show much more stable results and seems to be much less sensitive to the choice of how we define ensembling weights. This insensitivity is shown by all TAF models reaching considerable performance, at least when comparing the meta feature and PoE based TAF models to their SGPT counterparts. We argue this is due to the fact of TAF not having to scale performance across the meta data as it simply does not care. In addition, by other experts voting with an acquisition equal to zero, the weighting scheme does not matter that much anymore. We conclude that both SGPT-R and TAF-R are the most competitive surrogates while still keeping in mind that TAF in general is the better model class.

In a final experiment considering the perfomance in SMBO, we want to compare TAF and SGPT-R to the most powerful methods from the state of the art. Given the results on SVM and on Weka, for the SVM meta data we will include both FMLP and MKL-GP, as they are the closes competitors to our surrogates. For Weka, we only include FMLP, as MKL-GP is based on learning a full Gaussian process which is simply too much for the

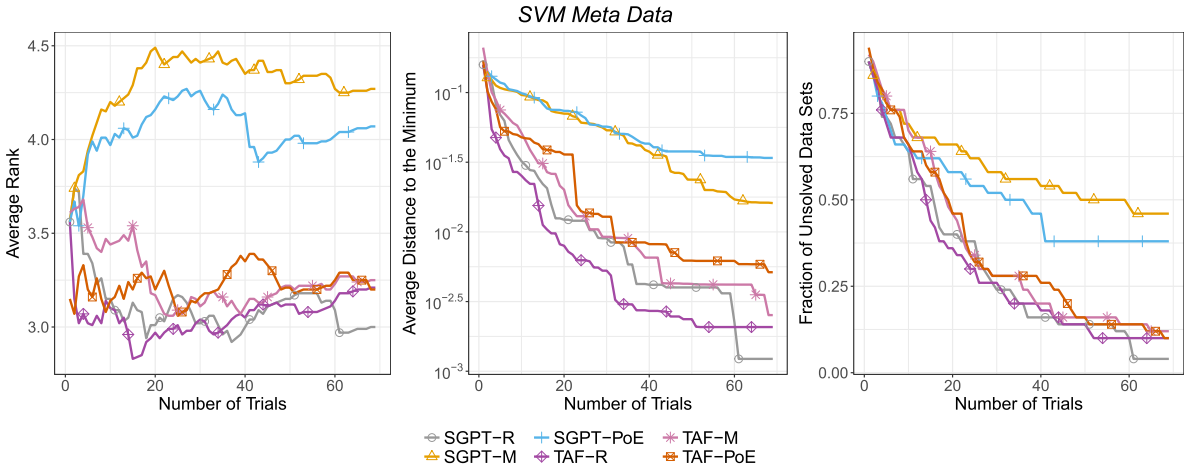

Figure 6.11: TAF and SGPT deliver similar competitive results on this meta-data set.

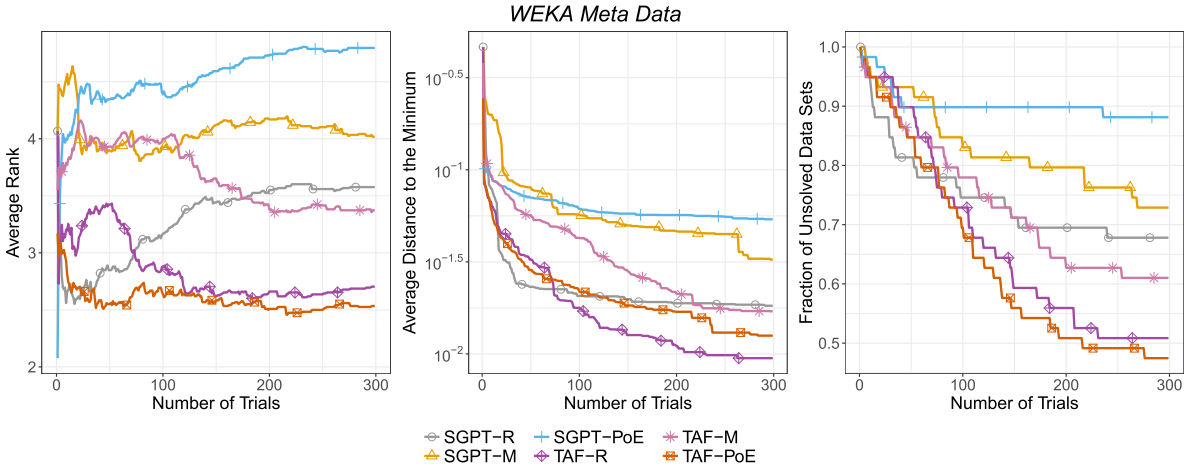

Figure 6.12: TAF provides a clear improvement over SGPT thanks to its adaptive use of meta-data and better way of dealing with different data set scales.

amount of meta data involved in the Weka experiment. Figure 6.13 shows the results on SVM for all three evaluation metrics, while Figure 6.14 shows the results on Weka.

For SVM, we see that SGPT-R and TAF-R fight a close battle for the number one which is likely won by TAF-R, however, this result depends on the evaluation metric one is interested in in a real-world application. While being more clear for TAF on the average distance to the minimum, the difference is rather marginal for the average rank and the fraction of unsolved data sets. FMLP fights the battle for third with MKL-GP, where FMLP seems to win the earlier trials but loses the later trials, again this depends on the evaluation metric one prefers. However, we see a clear lift brought about by our surrogate models SGPT-R and our transfer acquisition function TAF.

For Weka the lift is still existent, but we again see the good performance of FMLP

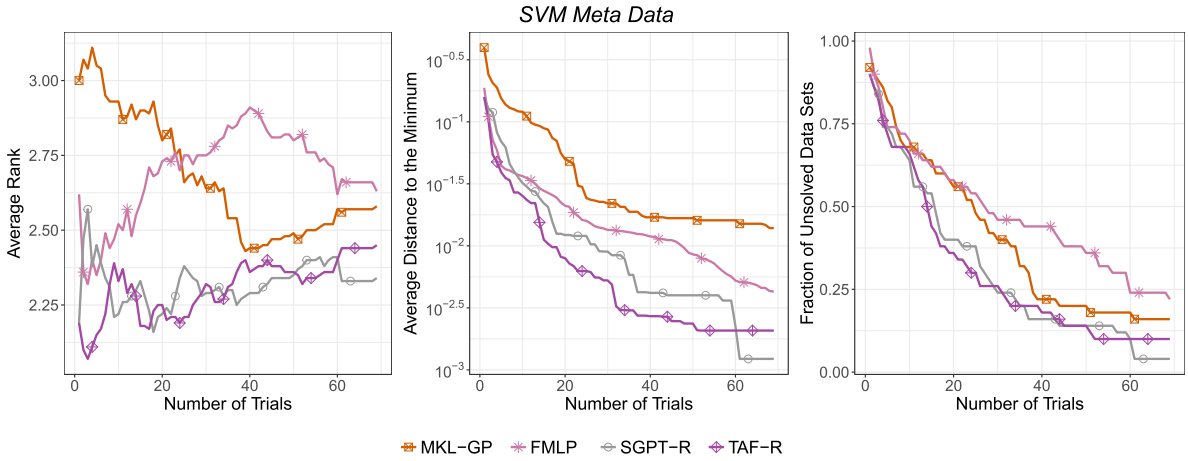

Figure 6.13: SGPT-R and TAF-R provide similar performances by outperforming the other competitors.

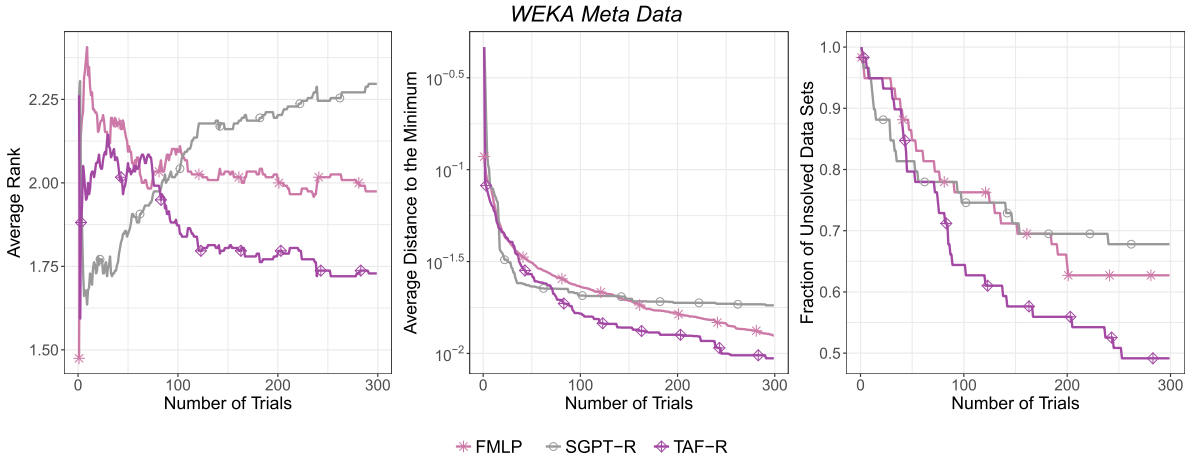

Figure 6.14: On this data set TAF-R is able show that it is more robust than SGPT-R.

after having observed the target response surface for a bit. It still beats SGPT-R over the time, but is outperformed by TAF, which simply shows more robus results than both of its competitors. FMLP wins on the very first trial, but quickly deteriorates, SGPT-R is ahead for a few trials, but also deteriorates afterwards even more heavily than FMLP. Unfortunately, results for MKL-GP are not accessible for a meta data set this large, but we would assume that it also does not outperform these models.

Finally, we conduct an experiment where we actually consider the time it takes for an evaluation of f within the evaluation metric. Previously, we have always assumed that one evaluation of f costs the same no matter which configuration we are querying. In practice, this is obviously not true, as for example more complex models require significantly more training time usually.

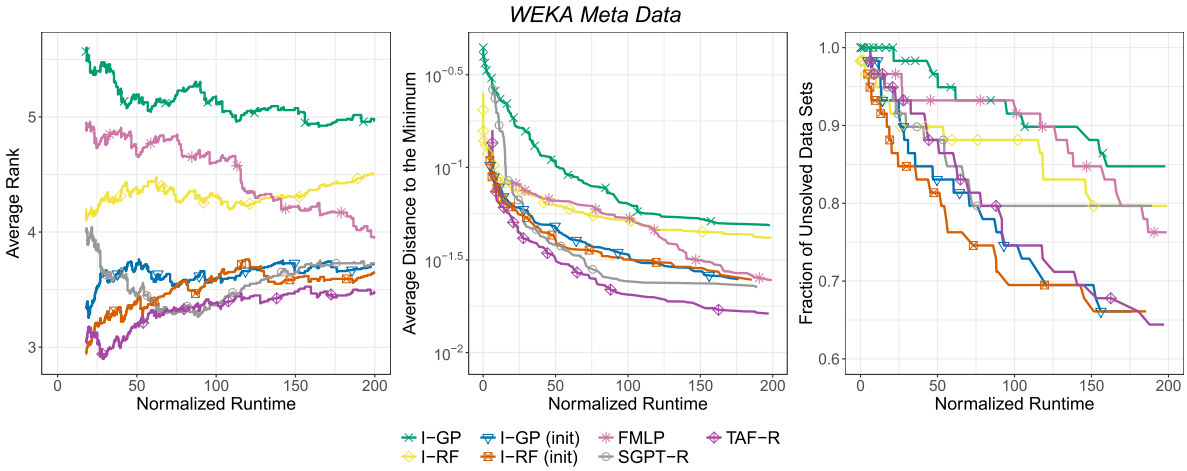

Figure 6.15: Comparison of TAF-R to its competitor method considering the run time. TAF-R still performs best. In contrast to the previous experiments, FMLP is worse and RF better.

We decided to add this experiment for the Weka data set, as it also considers model choice as hyperparameter and therefore the variance between learning times is much larger than for example for the SVN data set. Figure 6.15 shows all three evaluation metrics with respect to normalized runtime, where for each data set, we have normalized the runtime of each configuration by dividing it by the average training time needed. The average rank starting time is the time point where all surrogates have at least observed one hyperparameter configuration on all data sets. The starting time for the other two metrics is chosen when at least one configuration is evaluated for all data sets. While these sound very similar, the starting time for average rank is different as it has to wait for all competing methods.

While TAF-R still is the best performing method, the tides have changed for the independent Gaussian process as well as for FMLP. The independent Gaussian process shows much better performance while FMLP loses performance drastically. FMLP tends to choose configurations that need more runtime, this finding will be assured in the next chapter when model choice will be discussed. However, both models are agnostic to the runtime of each configuration, which makes the interpretation of the results challenging. Clearly finding surrogate models that are capable of knowing how much time budget they are offered, is a direction for future research.

Overall, we conclude that the surrogate models proposed in this chapter - especially TAF-R and SGPT-R - are quite powerful when it comes to pure performance. We want to also assess the statistical significance of these results, therefore we continue with those

experiments in the next subsection.

6.4.2 Significance Analysis

In this subsection, we will conduct two statistical tests on the performance of our surrogate models proposed in this chapter as well as their competitors. At first, we conduct a Friedman test [27] [28] with the corresponding post-hoc test being the two-tailed Nemenyi test [66] on both the SVM and Weka meta data at a specified number of trials. Afterwards, we execute a Bayesian hierarchical test [14] again on both meta data sets, however the Bayesian hierarchical test only considers two competitors, but will deliver a result for each trial. This way we do not make a single significance statement, but directly compare two competitors over the course of all trials. For more details to the tests that we chose, we refer the reader to the corresponding literature cited in the bibliography.W

The results for the two-tailed Nemenyi test conducted on the average ranks are presented in Figure 6.16 for SVM and in Figure 6.17 for Weka. We reject the null hypothesis of two methods performing not significantly different with respect to their average rank for a significance level of $\alpha = 0.05$, which is commonly used in statistical testing. For SVM, we show one comparison after the first trial on the left, and after the 30-th trial on the right. After more than thirty trials there is no difference in significances anymore, which is why we leave out plots further in SMBO. For Weka, we show the results after trial 30 on the left, while for trial 200 on the right.

Let us first have a look at the SVM results given in Figure 6.16, where methods are significantly different, if their difference in the average rank is above 1.7. For the first trial, we see that TAF-R is as significant as SGPT-R and only shares this position with FMLP, which also works remarkably well in our perfomance experiments, especially for the very first trial, or after having seen quite some fraction of the hyperparameter response surface. Furthermore, for the first trial, FMLP is as significant as MKL-GP, which has the same significance as some less powerful methods. The optimal method of always choosing the best hyperparameter on every data set is far away from all surrogates which makes a lot of sense as otherwise the problem would not be challenging.

Observing the significance after 30 trials, we see that now things have changed to a certain amount. Starting on the rather weak side, SCoT, I-GP and I-RF have gained some margin with respect to the average rank of Random, which makes a lot of sense. In the first trial random effects are much more prominent than in the later trials where surrogates really make sense of the data. In the middle of the diagram we see that most

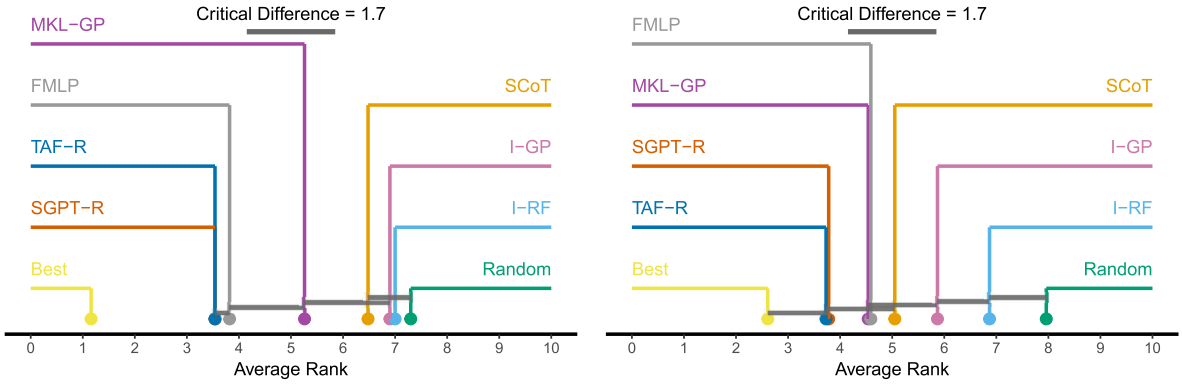

Figure 6.16: Comparison of all optimizers against each other with the two-tailed Nemenyi test on the SVM meta-data set. Groups that are not significantly different (at $p = 0.05$) are connected. The left plot shows the results after the first trial, the right plot after the 30th trial. After 30 trials there is no significant difference between our proposed methods TAF and SGPT and the potentially best method.

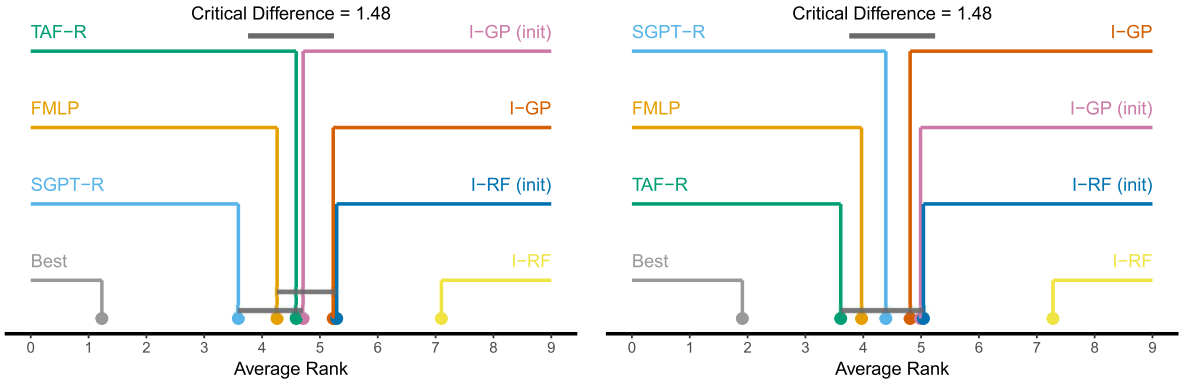

Figure 6.17: Comparison of all optimizers against each other with the two-tailed Nemenyi test on the WEKA meta-data set. Groups that are not significantly different (at $p = 0.05$) are connected. The left plot shows the results after 30 trials, the right plot after 200 trials.

of the surrogate models have moved closer together, which inherently means that the differences become less significant overall. However, on the positive side we see that now both of our proposed tuning strategies are not significantly different to the optimal tuning strategy, which is good to see after having observed around 42% of the target's response surface. FMLP, however, deteriorates in performance and groups with MKL-GP with respect to the significance of their performance.

Figure 6.17 shows the significance results on the Weka meta data, where the critical difference is 1.48 for a significance level of again $\alpha = 0.05$. After thirty trials, the best surrogate is SGPT-R, but it does not significantly outpeform FMLP, TAF-R and the

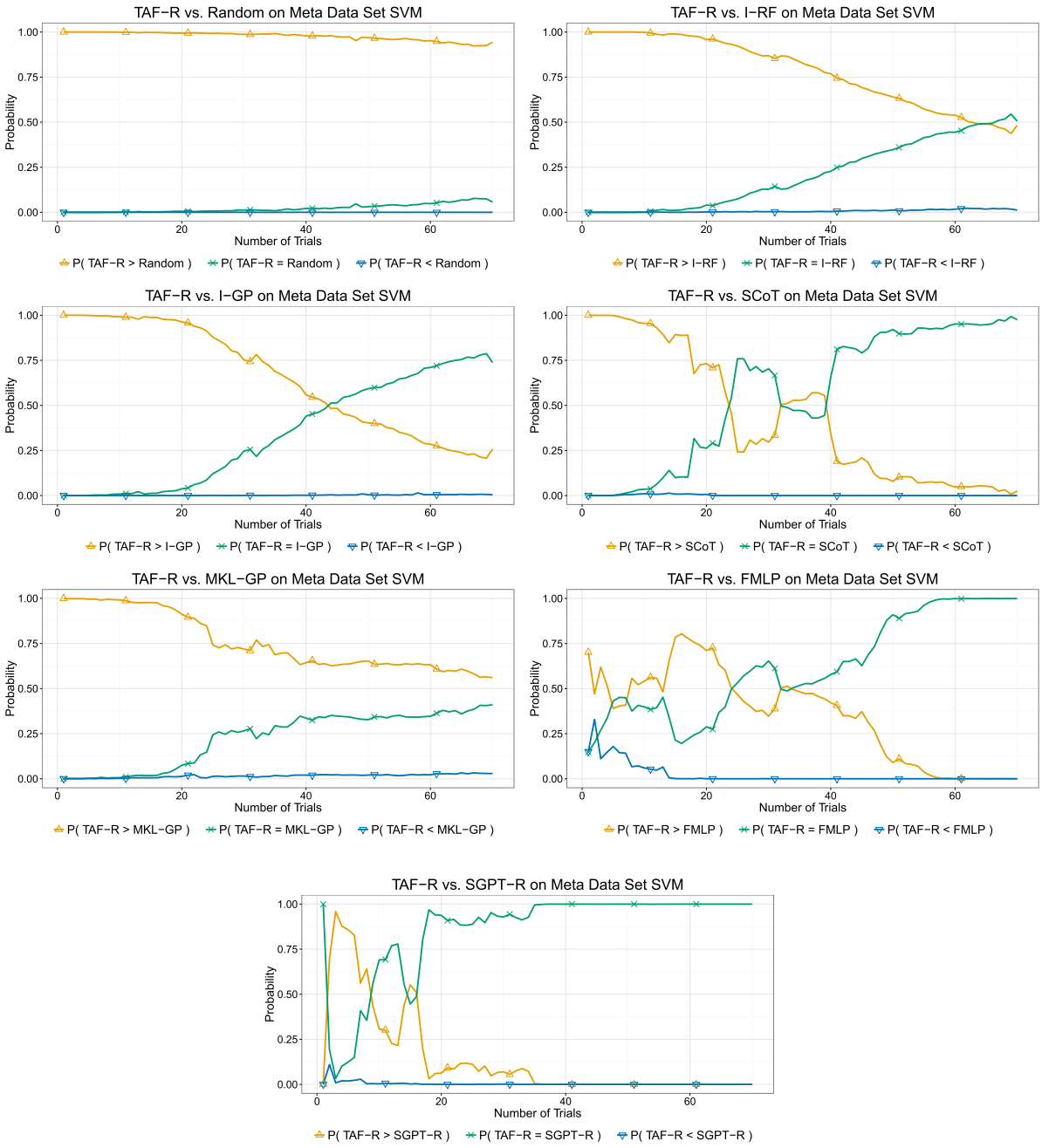

Figure 6.18: Comparison of TAF-R against all other optimizers using the Bayesian hierarchical test on the SVM meta-data set. Probabilities above 95% are significant. With increasing number of trials the probability for equality of both methods increases due to a limited number of considered configurations.

independent Gaussian process learned with initalization. The optimal surrogate is significantly better than all other methods, while the independent random forest is significantly worse than all other competitors.

After having conducted 200 trials, all methods move closer together, which is expected. Significantly different results are not given anymore, except for the optimal strategy still being significantly better than the rest and the independent random forest being significantly worse compared to its competitors.

In addition to the two-tailed Nemenyi test, we also conduct a Bayesian hierarchical test that is presented in [14]. The test compares two surrogate models Ψ_1 and Ψ_2 over the course of sequential model-based optimization, where for each trial, three probabilities are being estimated. The first value is the probability whether $\Psi_1 > \Psi_2$, where $\Psi_1 > \Psi_2$ means that Ψ_1 performs significantly better than Ψ_2 The second value estimates the probability of $\Psi_1 = \Psi_2$, of both surrogates performing insignificantly different while the last value describes the probability of $\Psi_1 < \Psi_2$. As proposed by the authors, we use a region of practical equivalence of $(-0.1, 0.1)$, meaning that two surrogates are not significantly different if the difference of their achieved accuracies lies in this interval. Certainly this region is applied on the standardized accuracy values.

For the Bayesian hierarchical test, we use again a significance level of $\alpha = 0.05$. However, since we estimate three different probabilities in contrast to the Nemenyi test, we can not only argue whether results are significantly different but also compute odd ratios of the probabilities, which allows us to compute how insignificant or almost-significant results are. As we can only conduct pairwise comparisons between surrogates, we consider comparing TAF-R to all other surrogates on both meta data sets, which already creates seven plots for SVM and six for Weka.

Figure 6.18 shows all three probabilities for comparing TAF-R to each other surrogate over the 70 trials conducted for SVM. We see that TAF-R is significantly better than Random, I-RF, I-GP, SCoT and MKL-GP. For SCoT and I-GP the probability of being equal rises considerably with more trials being conducted. It even surpasses the probability of TAF being better. In comparison to FMLP and SGPT-R, TAF still seems to be the better model, but the probability of both working equally well increases very fast. Overall, TAF-R seems to be significantly better or at least as good as the other models.

The results on Weka shown in Figure 6.19 show more or less the same results, where again the amount of significance overall is less than for SVM. TAF-R beats both random forest surrogates as well as the independent GP, but for the GP that is initialized with meta knowledge, the difference is already much smaller. The probability of both

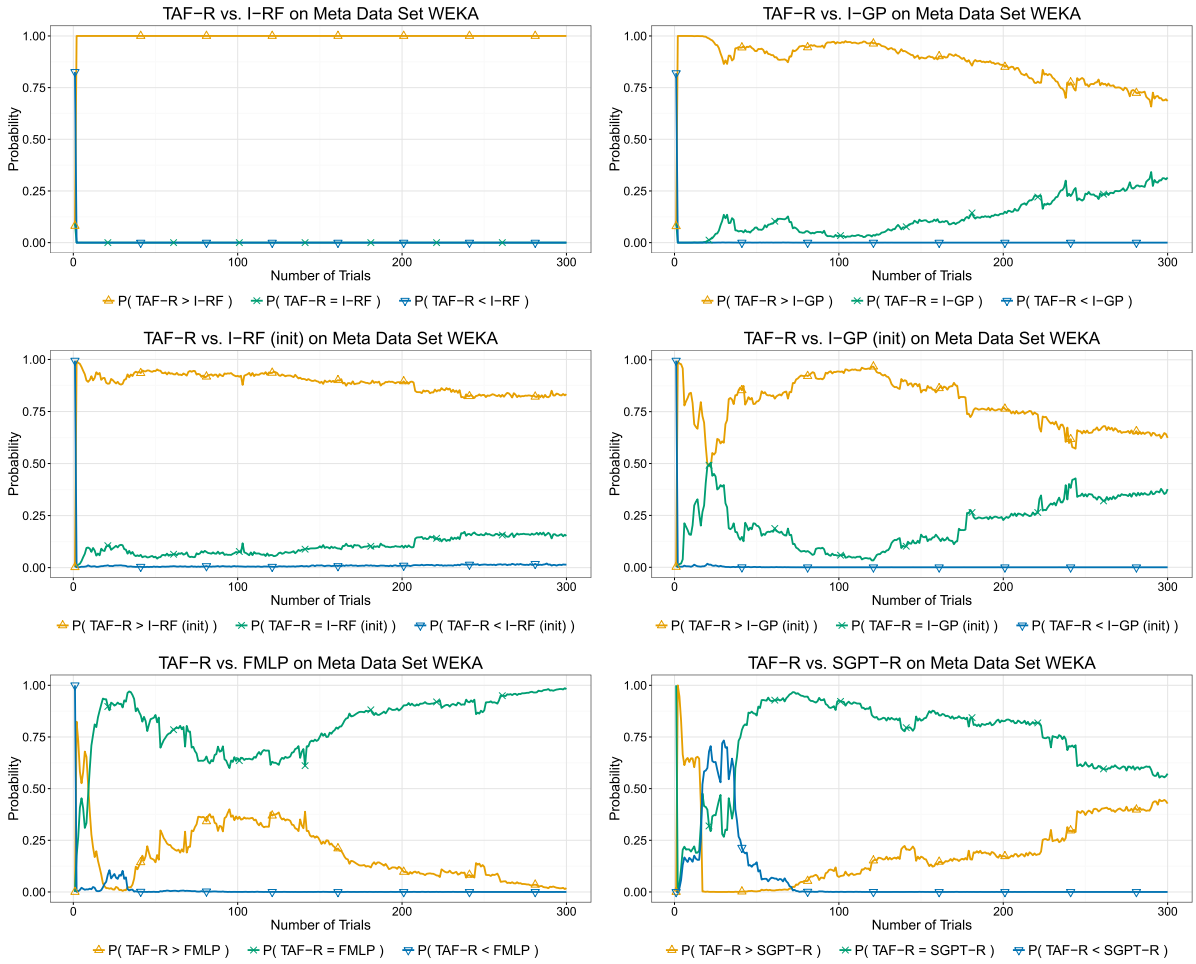

Figure 6.19: Comparison of TAF-R against all other optimizers using the Bayesian hierarchical test on the WEKA meta-data set. Probabilities above 95% are significant. With increasing number of trials the probability for equality of both methods increases due to a limited number of considered configurations.

surrogates performing equally well is considerably high over the different trials. When comparing TAF-R to FMLP and SGPT-R, the probability of both methods being equal dominates both other probabilities. While being less significant, we still argue that TAF overall is the best surrogate, as it either performs better or equal to its competitors.

6.4.3 Runtime

Finally, we conduct an experiment taking into account the runtime of learning the different surrogate models on the meta knowledge. As we have discussed, a single Gaussian process has a training time of $\mathcal{O}(M^3n^3)$, where n is the number of hyperparameter configurations observed in the meta data, and M is the number of different problems

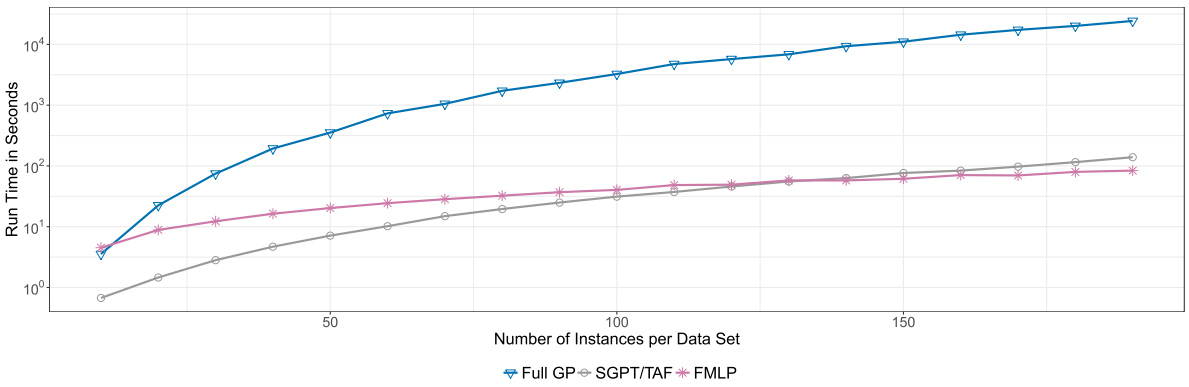

Figure 6.20: SGPT (and hence also TAF) is clearly outperforming a single Gaussian process on the full meta-data with respect to scalability. FMLP, which is based on a neural network, has a linear training time and provides faster results for larger data sets.

considered. By learning a product of experts we can reduce the training time to $\mathcal{O}(Mn^3)$, which is a considerable reduction if a single GP can still be learned relatively fast on n many instances. However, if we ever face the problem where the hyperparameter grid for a single data set is already too large, we can consider dividing the grid into subgrids and learning more than one expert per data set. For example, for SVM we could consider learning one expert for each of the three different kernels.

In order to show the scalability properties of our approach, we have created an artifical meta data set which consists of $M = 50$ many problems. We vary the size of the response surface n from 10 to 190, and measure the training time to learn the surrogate model on all of the meta knowledge. We compare SGPT (or TAF, as both have the same training time) with FMLP and learning a full Gaussian process. Results are given in Figure 6.20, where on the x-axis different values of n are plotted and the y-axis shows the training time in seconds on a logarithmic scale. When looking at the results we immediately see that learning a full GP clearly takes most time, where the difference is really big for large n. For $n = 190$, learning a full GP takes roughly seven hours, whereas learning a product of experts only takes two minutes. FMLP has a linear learning time that is dependent on hyperparameters such as how many epochs one wants to conduct, where we have used the settings proposed by the authors. For small datasets, SGPT is learned faster than FMLP, however, this advantage is lost the bigger n becomes. At some point, FMLP is learned faster than SGPT, nevertheless, both run times are still very similar.

Our main goal was to build surrogate models that deliver powerful posteriors, do not

use many hyperparameters, and can be learned efficiently. By proposing SGPT as well as TAF, we believe that we have made a big step towards reaching this goal. Likewise, observing that FMLP - the surrogate model that was presented in the previous chapter - still delivers quite competitive performance validates the approaches of this thesis.

CHAPTER 7

Hyperparameter Optimization Across Problem Instances

In this chapter, we will use the surrogate models that we have presented in the previous chapters and show how they perform when they are learning across problem instances other than different data sets. By talking about problem instances we think of things such as learning not only the hyperparameter performance but model choice as well. Additionally, we can think of generalizing across different problem tasks, for example by transfering knowledge about hyperparameter performance from a model on a regression task, to the same model being applied for a classification task. This appears to be possible, at least if the model allows to be used for both problems. Furthermore, one could think of generalizing hyperparameter performance across different preprocessings of the data or using different loss functions such as using pointwise, pairwise or even listwise loss functions in a ranking task.

We have covered two of these instances of hyperparameter optimization across problem instances. At first, we will show how a joint hyperparameter optimization and model choice can be performed simply by building meta data that covers more models and applying the usual Bayesian optimization scheme while using a powerful surrogate. Afterwards, we present results on learning neural networks for a regression and (multi) class classification tasks, where we seek to share hyperparameter knowledge across tasks.

7.1 Combined Hyperparameter Optimization and Model Choice

In this section we discuss the results of our work in [79], where we jointly optimize hyperparameters and model choice. As a side note, this work has been published in 2015 which was before the work of the previous chapter where we already learned hyperparameter performance across models. We present the results of some surrogate models used on the Weka data set which we have introduced in section 4.1.3. The meta data consists of roughly 1.3 million experiments over a total of 59 data sets using a total of 19 different classifiers among SVMs, MLPs, logistic Regression, but also simple classifiers such as naive Bayes or decision stumps.

One question that immediately springs to mind is how to define meta features for different models, which is quite hard to answer. At first, one could think of defining a meta feature that measures the complexity of a model, by measuring the number of parameters employed, however, this depends on the data set that is to be solved and therefore does not make much sense. Staying with that thought, one could for example define whether a classifier is linear or not, which in isolation is quite rudimentary. Other meta features could be defined from the learning algorithm, for instance SVMs are usually learned with SMO, decision trees are learned greedily and other parametrized models are usually learned with gradient descent. Nevertheless, we would at least have to learn latent characteristics with all optimization algorithms in order to make use of the given information. We could also distinguish between generative and discriminative models, but since the task at hand is only to optimize hyperparameters for a supervised learning problem, discriminative models are expected to work better anyway.

However, as it turns out, we seem to not need meta features for different models as different models can simply be distinguished by their hyperparameters, as hyperparameters are usually - of course not always - partly disjoint for different models. In this way, the surrogate automatically knows for which model it currently is predicting hyperparameter performance, simply by the non-zero feature values in the meta instance. In general, this is obviously not true, as different models may be learned with gradient based optimization and therefore some form of step size and its control has to be defined prior to learning.

Additionally, we can think of regularization weights which define the amount of regularization that is inflicted on the model parameters to keep them from overfitting. On the one hand, by treating such hyperparameters individually for each model, we

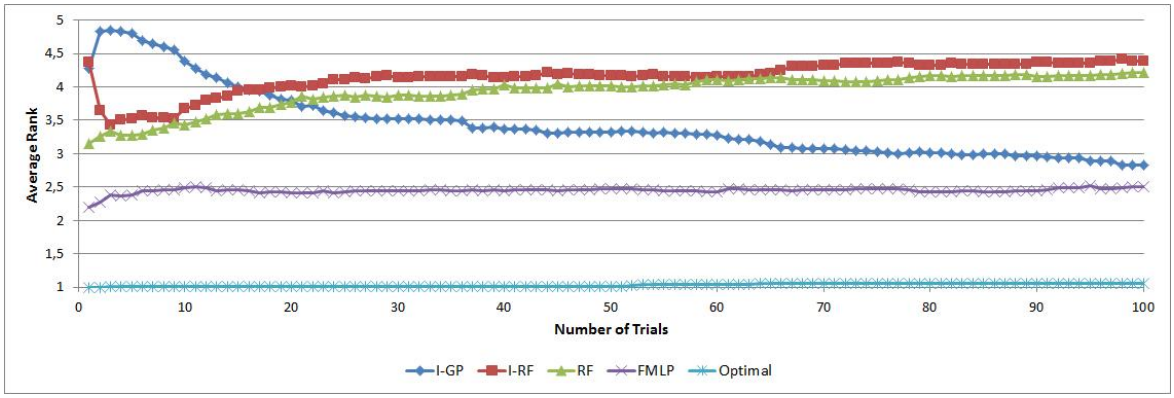

Figure 7.1: Average Rank across all competing methods. Besides the artifical Optimal strategy, FMLP clearly outperforms all other surrogates in the first 100 trials.

somehow follow an easy way out of the problem, as then all the surrogates and Bayesian optimization works out of the box. On the other hand, we neglect that these hyperparameters are actually doing the same thing for different models and therefore do not transfer knowledge of how a hyperparameter performs for one model to how it performs for the other.

7.1.1 Experimental Results

In our initial work, we have learned four different surrogate models, firstly we learn an FMLP as presented in Chapter 5. Secondly, we learn a random forest as was proposed by Frank Hutter in his seminal work [39]. Lastly, we assess how these two surrogates perform against models that are not learned across data sets, by also evaluating an independent Gaussian process as well as an independent random forest. Other competing methods, such as SCOT and MKLGP were impossible to run as both do not scale to large amounts of meta data, which we are given in the weka meta data. The random surrogate was not evaluated, simply because the number of hyperparameter configurations for each model is not equal. Out of this reason, a random surrogate principally oversamples those models that have a larger configuration space, and therefore is not random anymore. As an outlook, Figure 7.3 in the next subsection shows the ratio that each model possesses in the whole hyperparameter space.

The average rank among all competitors for a fixed number of 100 trials can be seen in Figure 7.1, where we see at first glance that FMLP outperforms all other methods. It always has the lowest average rank lying roughly between 2 and 2.5, and therefore the best performance of all methods. However, as other models start learning more, the

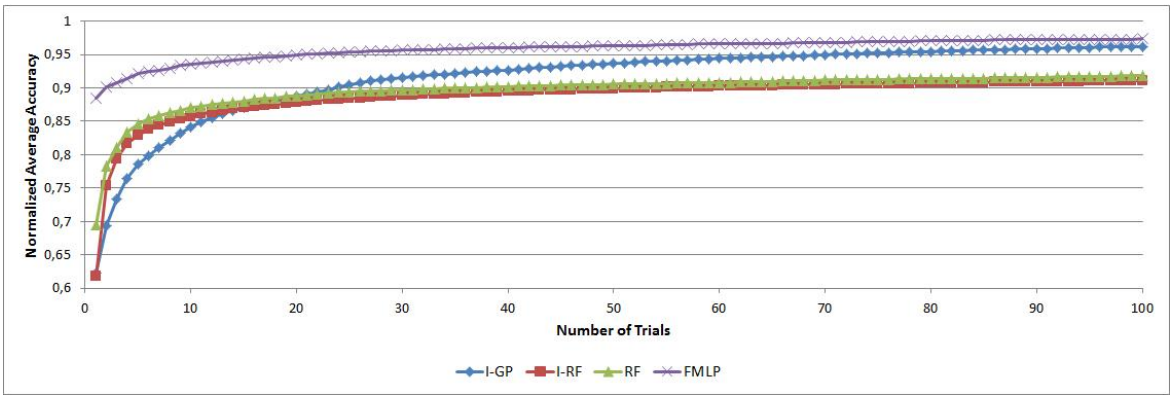

Figure 7.2: Average Normalized Accuracy of each competitor. FMLP starts at an average accuracy of 0.9 which is quite remarkable.

average rank of FMLP increases a little bit, this effect can be observed in the first few trials of the optimization. The random forest that has access to the whole meta data is the second best performing surrogate, at least for the first 20 trials. We can also observe that the average rank of this method is also lower than the average rank of the random forest without meta knowledge, although the difference seems to be quite small after a few iterations.

What is most surprising is the performance of the Gaussian process. Excluding the very first measurement - as it is due to random effects for the independent models - the Gaussian process seems to behave worst. However, after 20 trials or so, it starts to overtake both random forest models, even without having any meta knowledge. Gaussian processes seem to adapt to the new data set really well, if they would not suffer from their bad initializations. This finding has been demonstrated in this thesis in many experiments, which is why we also argue that Gaussian processes are principally a very good model for our task, and why the models from the previous section seem to perform best.

Another perspective of the same results is given in Figure 7.2 where we plot the average normalized accuracy for the first 100 trials. The performances of the different surrogate models are the same as for the average rank, we again see that FMLP has the highest accuracy over all trials. Additionally, we see the small difference between the random forest with and without meta knowledge, as well as the bad performance of the GP who is then able to catch up.

What is remarkable is a glance to the y-axis, where the average accuracy is plotted. We can observe that FMLP starts at an average accuracy of 0.9, meaning that on average it finds 90% of the maximal accuracy found on the whole grid in the first trial, which

is quite marvellous. Moreover, after having conducted 100 trials, FMLP attains an 97.5% of the maximal accuracy on average after having evaluated only 0.005% of the hyperparameter grid. Only the cold-start issue for the latent factors of the target data set still remains, but somehow also seems to be mitigated by having learned decent latent features for all other entities.

Comparing this to the other methods, we can see that a random forest with meta knowledge starts at an average accuracy of almost 0.7, which is much smaller. The independent methods deliver an accuracy of roughly 0.6, which is the accuracy that one obtains by simply choosing a configuration at random. At this point, please note that this value does not have to be 0.5, as hyperparameter performance is in no way uniformly distributed. This means that we may have data sets where almost every hyperparameter configuration leads to good results or conversely, one data set where almost none of the configurations works well.

7.1.2 Model Choice

After having looked at the results on a performance based level, we also want to shed light on the models that are being chosen, especially by our proposed surrogate model. Figure 7.3 shows the model choice of an SMBO run with FMLP on the weka data set. For each of the 19 models, we have plotted both their ratio in the hyperparameter space as well as their ratio in how many times FMLP evaluates hyperparameters of that specific model. The white bars can be understood as the probability of a random surrogate choosing a hyperparameter configuration of that very model. As we can see, MLP and SMO (support vector machines) dominate the hyperparameter space. If we had included a random surrogate in the previous experiments, it probably would have performed well, as it oversamples exactly the more powerful models.

For FMLP, we can observe that it also favors the complex models such as MLPs, random forests and support vector machines over the less complex models such as naive Bayes and logistic regression. Moreover, even weaker models such as decision stumps - which are effectively decision trees with only one split - are not even considered at all. Additionally, ZeroR - the name means zero rules - is also not considered, as it is a classifier that simply predicts the majority class. Our surrogate has learned that these models are very weak on the other data sets and therefore ignores them, which is an expected result. For all other models it seems that testing them a few times on the new data plus having the meta knowledge available seems to be enough to judge whether to further explore using this model or not. Automatically gathering this knowledge of

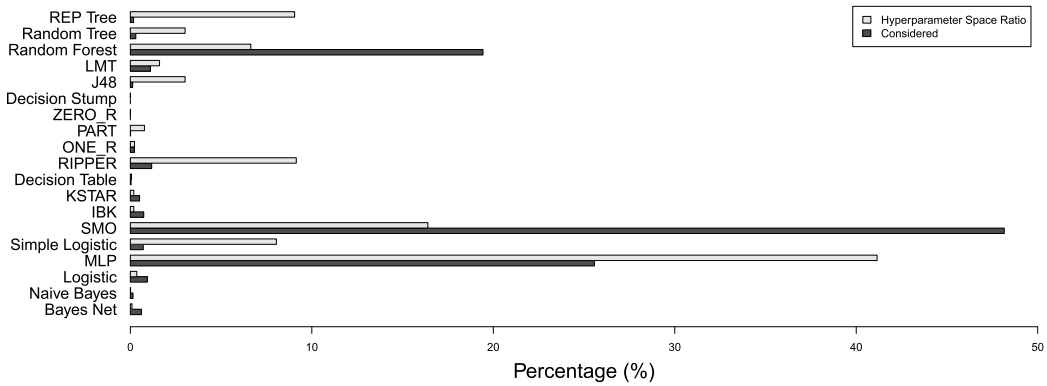

Figure 7.3: An overview of each model's share in the hyperparameter space as well as the ratio of how many times FMLP picks a model in the first 100 trials. FMLP systematically oversamples the strong classifiers such as SVMs, Random Forests and additionally tries many configurations of Multilayer Perceptrons. Other model classes, such as logistic regression are only evaluated a few times. Weak learners such as decision stumps are not considered at all.

well performing models and hyperparameters is exactly what experienced data scientists do. Having found a way to facilitate this process to speed up hyperparameter search is precisely what we wanted to do and therefore this result seems quite rewarding.

7.2 Hyperparameter Optimization Across Problem Tasks

In this section we will present the results from our work [77] which is currently being reviewed in the Archives of Data Science. In the paper *Hyperparameter Optimization Across Problem Tasks* we have investigated the problem of learning hyperparameter performance across different kind of problems, such as classification and regression tasks. To accomplish this goal, we conduct experiments that assess how well different surrogates can learn across tasks on our neural network meta data set, which was presented in Chapter 4.

We will conduct two experiments. At first, we will use the whole meta knowledge to predict for a data set that has been left out, simply to evaluate the overall performance of different surrogate models on this data set. Secondly, we will restrict the surrogate's training data to one task only and compare the performance of this surrogate to the one using all of the meta knowledge. As an example, we compare learning a surrogate only

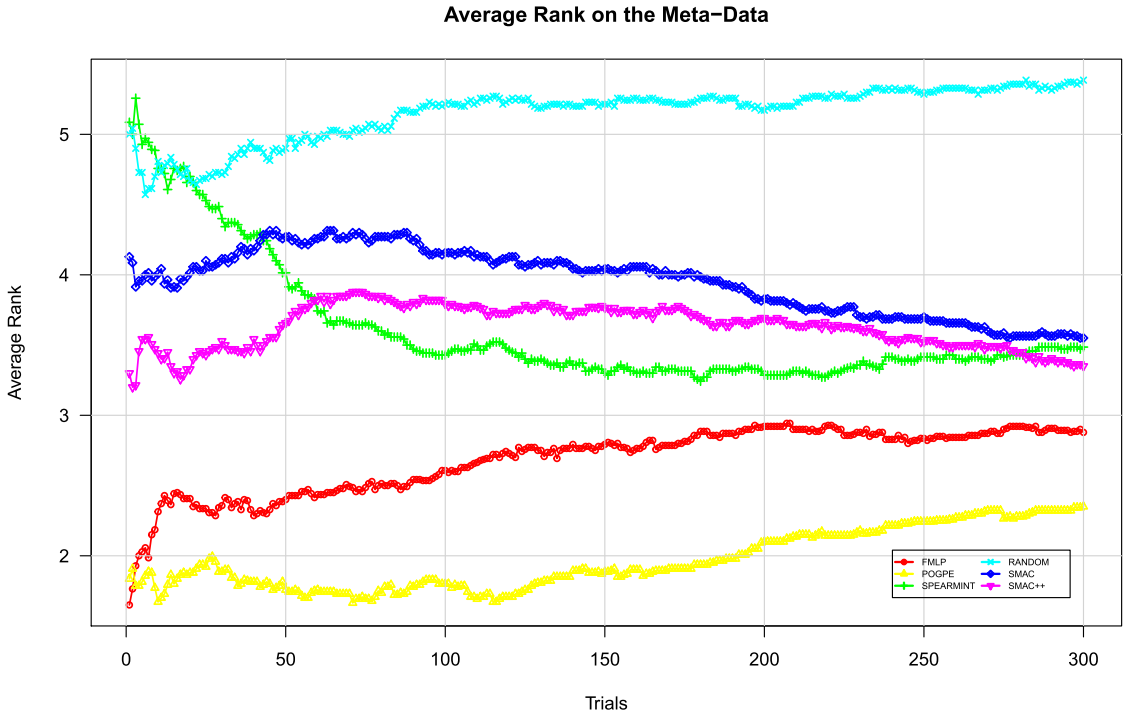

Figure 7.4: Average Rank among all competing methods on the full meta data set, including regression, binary classification and multiclass classification tasks. The lower the average rank, the better.

on regression data sets to then predict for regression problems with the same surrogate being learned on the regression meta data but additionally being also learned on the classification meta data.

7.2.1 Hyperparameter Optimization on All Tasks

For the first experiment, we proceed as follows. For each target data set, we learn all surrogates on the remaining 69 data sets and then perform 300 trials of sequential model-based optimization, which resembles a good 10% of the hyperparameter grid. Figure 7.4 shows the average rank of all competing methods for all 300 trials conducted. As usual, we observe the very poor performance of the random surrogate model, however, a single Gaussian process seems to also act randomly in the beginning, to then slowly catch up to some of the other baselines around trial number 50. SMAC has a worse performance than its counterpart that learns across problems and data sets, the distance of both approaches is bigger in the beginning and then decreases over time, which is natural as meta knowledge may not be that necessary after some parts of the response

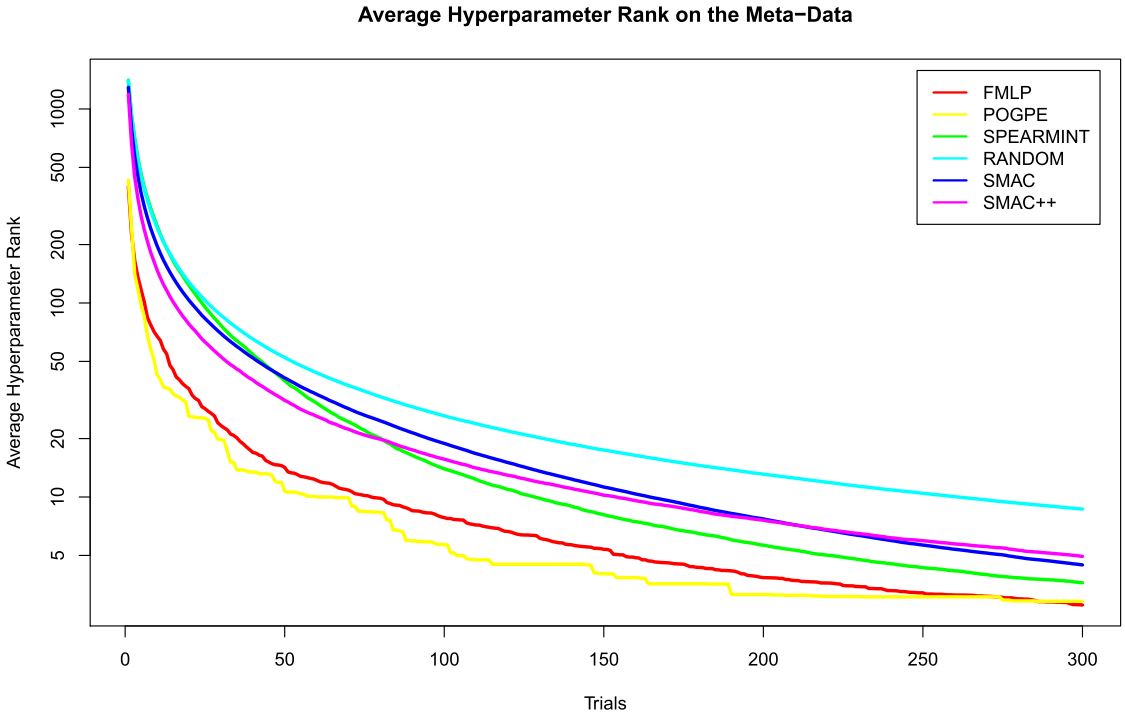

Figure 7.5: Average Hyperparameter Rank of all competing methods on the full meta data set, including regression, binary classification and multiclass classification tasks. The lower the average hyperparameter rank, the better.

surface have been oserved. However, at least for 300 trials SMAC does not catch up with SMAC++. The best performing methods are FMLP and POGPE, where the former method has a better start in the first two trials. Unfortunately for FMLP, its performance deteriorates quickly, at least in comparison to POGPE.

More or less the same results can be observed in Figure 7.5, where the average hyperparameter rank is plotted for 300 trials. POGPE and FMLP have a much better start, as their first hyperparameter configuration has an average rank of roughly 300. The other methods start somewhere around 1,500. The difference between FMLP and POGPE seems smaller than in the average rank, this is likely due to the fact that even the slightest difference in validation performance impacts the average rank, whereas the average hyperparameter ranks may still be similar. Overall, the performance of FMLP and POGPE is quite satisfactory.

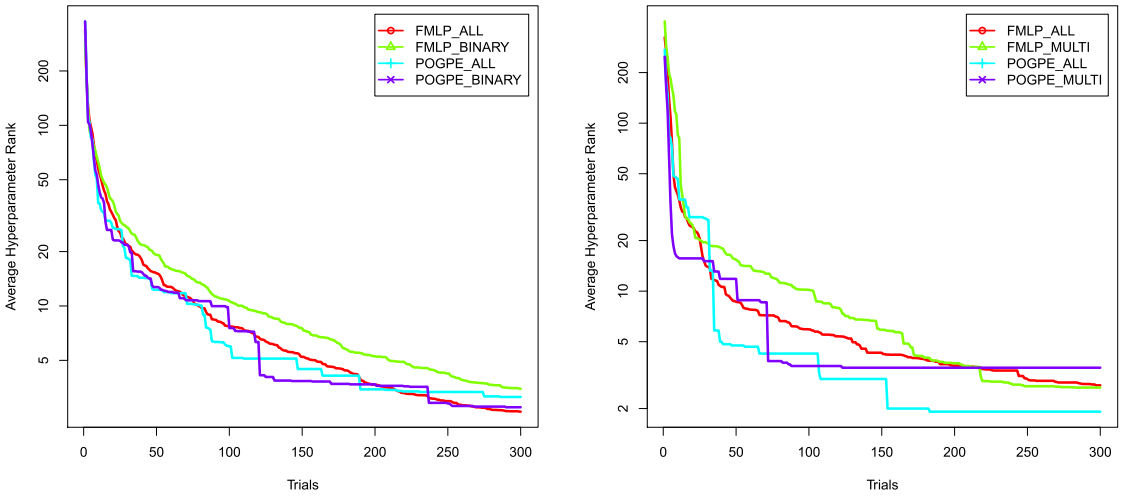

Figure 7.6: Average hyperparameter rank for binary and multiclass classification tasks.

7.2.2 Transfering Hyperparameter Performance Across Tasks

In this experiment, we will evaluate the influence of having additional meta data of different tasks. In order to do so, we learn surrogate models on the meta knowledge of one task only, say for the task of binary classification. Experiments are then done in a leave-one-out fashion, where always one of the binary classification data sets will be considered and the results will be averaged across all folds as usual. As a second experiment, we include also the meta information from other tasks, to then assess whether there is a lift in the performance of the hyperparameter configurations chosen. This experiment only makes sense for surrogate models that are able to learn across tasks, therefore we have chosen to only evaluate POGPE and FMLP, since they were the dominant models on the experiments including all the meta data.

Figure 7.6 shows the difference in average hyperparameter ranks between including meta data from other tasks to learning solely on the target task. Results are shown for both classification tasks, where we differentiate between binary classification on the left and multiclass classification on the right. The results are mixed, especially between both surrogate models. Where FMLP seems to learn additional information about hyperparameter performance, POGPE fails to do so, or the benefit is not that clear. For binary classification, including also multiclass and regression tasks does not seem to help, as both curves are roughly overlapping, there is only a difference between trial 70 to trial 180, with both surrogates being better for some trials. FMLP has a consistent lift

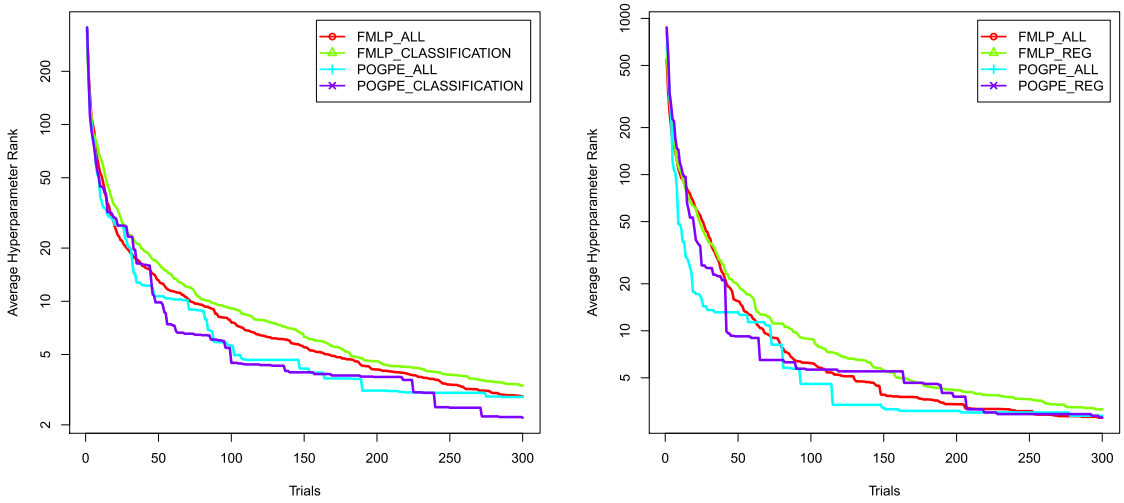

Figure 7.7: Average Hyperparameter Rank for the joint classification task (left) and for regression (right). For the joint classification, the inclusion of the regression meta data has no huge impact, surprisingly also for regression the impact is not too large.

as the red curve is strictly below the green curve, we argue that this is due to the latent feature embedding that is learned by FMLP.

For multiclass classification, the picture is somehow similar, there still seems to be a principal lift for FMLP, although the curve is not strictly below the FMLP variant that is trained only on the multiclass classification data sets. In contrast to binary classification, for POGPE the inclusion of more meta data seems to have a more fundamental lift, however, not immediately in the beginning of SMBO. We argue that the lift is more fundamental, simply because the multiclass classification meta knowledge is much smaller than the binary classification meta knowledge.

This finding can also be seen in Figure 7.7, where again the average hyperparameter rank is plotted, this time for the joint classification task and the regression task. By joint classification, we merge both binary and multiclass classification tasks. On the left, we see that FMLP still shows a lift, but the difference is quite small, which is due to the amount of meta knowledge that is already contained in the classification task. As only a few regression meta data sets are additionally included, it is reasonable that there is no huge lift in the performance.

For the regression task, the story should be different, simply because there is not much meta knowledge about regression tasks. FMLP still shows a principled lift, but the difference is quite small and only shows after roughly 50 trials, which is a little

disappointing. For POGPE the lift is also quite small, at least it is consistent at the beginning of the SMBO iterations, which usually is the most interesting region.

Overall, we conclude that using additional information likely helps, but the impact is different depending firstly on how much meta data one already has, and secondly on the surrogate model. FMLP seems to make more use of the additional data, however, since it is still a little worse than POGPE, it is questionable how useful the performance improvement is. One way of marrying both may be to use an ensemble surrogate, which likely combines the advantages of both.

CHAPTER 8

CONCLUSION

In this thesis, we have seen two main approaches for solving the problem of hyperparameter optimization or more generally algorithm configuration in machine learning problems. At first, we have viewed different data sets as distinct users that interact with models, learning algorithms and preprocessing in a relational way. This allows us to estimate latent characteristics of all involved entities directly using a factorization-based approach. Secondly, we have discussed scalable surrogate models for hyperparameter optimization, which are able to scale to large amounts of meta information, while still showing the important properties of Gaussian processes. Finally, we have learned hyperparameter performance across different tasks such as regression and classification, as well as for categorical decisions such as model choice. Overall, we see that our approaches work well in comparison to the state of the art at the time of publication, however, there is still a lot of room for future work that has already been discussed throughout this thesis.

The relational approach using factorization models in a surrogate lessens the dependency on meta features, which still are not understood in depth. It is not clear which meta feature helps in which scenario, therefore simply using many meta features and hoping their amount makes them expressive is currently the way to go. Additionally, as meta features are used to allow the surrogate model to distinguish between data sets, the decision of simply taking many of them is supported. Since these meta features can be viewed as hyperparameters that have a strong influence, their interaction with the validation loss on the target data needs to be estimated. Learning these characteristics

automatically in a supervized way, as we have proposed in this thesis, is a way to solve the meta feature problem.

Interestingly, also the scalable surrogate models that we proposed alleviate this issue. At least given our current division of the whole meta data into meta data of each problem, the models based on products of experts do not need meta features anymore, since each expert learns only one response surface. On top of that, meta features are also not needed to compute the ensembling weights, as the ranking approach based on the actual response surfaces seems to be a better distance between data sets, at least for our problem of hyperparameter optimization.

The models TAF and SGPT seem to unite the best of what is possible. At first, they are easily scalable to large amounts of meta knowledge, which was our key motivation. Secondly, optimizing their hyperparameters can be done in an automatic fashion. Finally, at least TAF solves the issues of response surfaces having different scales in a very elegant way by combining the acquisition functions of each expert.

Nevertheless, there are many directions for future work. First of all, the surrogate models we are proposing are agnostic to the time budget that they have, the decisions that are made are based only on the history of observed configurations, but do not consider how much time is left. Naturally, we would assume that a decent strategy decides to explore more in the beginning, to then try to heavily exploit the surrogate if time is running out. To model this, surrogates must be aware of the time budget. We have tried several ideas in this direction, for example using a weighted expected improvement, where the uncertainty is upweighted in the beginning and downweighted in the end, but sadly this has not shown any significant difference.

Reinforcement learning is an area where optimal policies of decisions are found given a time budget and a reward function, that rewards (or punishes) decisions. This sounds exactly like the methods we are searching for; however, reinforcement learning usually is applied in areas where simulation is easily possible. For hyperparameter optimization, an evaluation of the validation performance is costly, which demands for clever ideas in modelling it as a reinforcement learning problem.

Another possible direction for future work is to include learning curves into the whole process, which is already done by related work. However, so far nobody seems to use meta knowledge, i.e. learning curves of other problems in order to speed up the identification of promising configurations.

Finally, optimizing hyperparameters for problems with more complex data such as learning deep neural networks for image classification or for time-series analysis is a

direction that seems to be en vogue. Here, one could also try to generalize validation performance across another problem instance: heavy preprocessing routines. We have discussed this, but not integrated an experiment where we for example try to learn the preprocessing for a speaker identification model from the already known performance of different preprocessings of a speech recognition task.

To conclude, there are still a lot of steps that need to be made in order to make machine learning work autonomously. However, we believe that the current hyperparameter optimization strategies are already a big support for data scientists and those users that want to apply machine learning to their problems, but do not know every detail of every model and learning algorithm. In this way, methods proposed in this thesis facilitate machine learning, increase its usability and therefore aid in spreading its usage.

APPENDIX A

APPENDIX

The appendix will be used to introduce multivariate Gaussian distributions and a few of their various properties. Especially interesting will be the conditional distributions, the marginal distributions of a subset of the random vector, as well as the product of Gaussians. The conditional distribution of multivariate Gaussians will lead to the main prediction formulas to compute mean and uncertainty of a Gaussian process, which is a commonly used surrogate in Bayesian optimization. The marginal distribution will be used to compute uncertainties of non-Gaussian models around a mode of the posterior. Finally, the product of n many Gaussians will give rise to the product of Gaussian process experts, which is a surrogate model that we proposed in this thesis.

A.1 Multivariate Gaussian Distributions

We say a variable $x \in \mathbb{R}^D$ is multivariate Gaussian distributed around mean $\mu \in \mathbb{R}^D$ with symmetric and positive definite covariance $K \in \mathbb{R}^{D \times D}$, if it has a probability density

$$p(x) = \frac{1}{\sqrt{(2\pi)^D \det(K)}} \exp\left(-\frac{1}{2}(x-\mu)^\top K^{-1}(x-\mu)\right) \quad . \tag{A.1}$$

Figure A.1 shows the density of a multivariate Gaussian defined on two variables. Usually, Σ is used for the covariance matrix, however, since we will use Gaussian processes which define their covariance through a positive definite kernel, we already denote the covariance matrix by K. In many scenarios, it makes sense to use a precision matrix

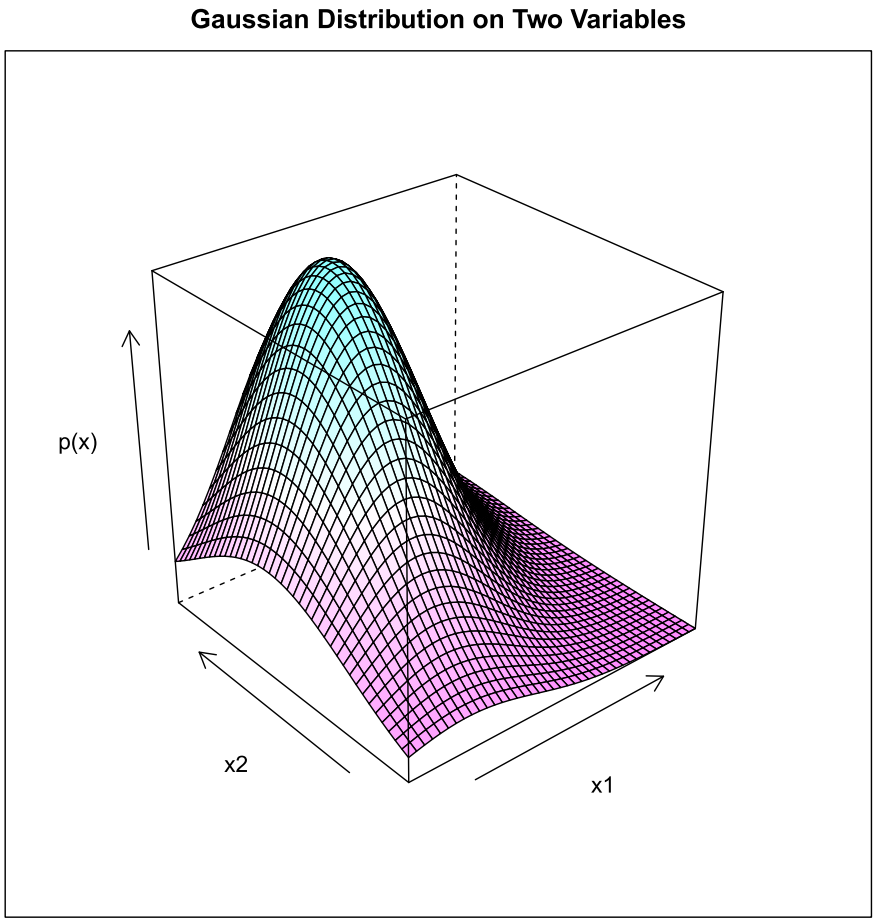

Figure A.1: A multivariate Gaussian defined on two variables, the z-axis shows the density of the variable $x = (x_1, x_2)$

P, which is the inverse of the covariance matrix, so $P = K^{-1}$ as this parametrization sometimes is simpler to handle.

In order to determine, whether a density is a Gaussian density, it suffices to check the terms in the exponential function in order to find a suitable mean and covariance. Expanding the term we can write it as

$$\begin{aligned} -\frac{1}{2}(x-\mu)^T K^{-1}(x-\mu) &= -\frac{1}{2}x^\top K^{-1}x + \frac{1}{2}\mu^\top K^{-1}x + \frac{1}{2}x^\top K^{-1}\mu - \frac{1}{2}\mu^\top K^{-1}\mu \\ &= -\frac{1}{2}x^\top K^{-1}x + x^\top K^{-1}\mu + C \end{aligned} \tag{A.2}$$

where we have used the symmetry of K^{-1} and the fact, that transposing a scalar returns the same scalar. The last term is independent of our variable x and therefore can be seen as a constant term that we multiply to the density.

The next property we show is the affine transformation of Gaussians, which is the Gaussian that results by multiplying the random variable X with a matrix A and adding a constant term b.

Lemma A.1. *Let $X \in \mathbb{R}^D$ be a random variable that is multivariate Gaussian distributed. Then, any affine transformation by a matrix $A \in \mathbb{R}^{L \times D}$ and a constant $b \in \mathbb{R}^L$ is also multivariate Gaussian distributed, if $AKA^\top$ is not singular.*

$$X \sim \mathcal{N}(\mu, K) \quad \Longrightarrow \quad \tilde{X} = AX + b \sim \mathcal{N}(A\mu + b, AKA^\top) \tag{A.3}$$

Proof. The proof is rather simple and straightforward. We only need to compute the mean and the covariance matrix of the resulting distribution. For the mean, we get:

$$\mathrm{E}(\tilde{X}) = \mathrm{E}(AX + b) = \mathrm{E}(AX) + \mathrm{E}(b) = A\mathrm{E}(X) + b = A\mu + b \tag{A.4}$$

as the expectation is linear and the expectation over b simply returns b. For the covariance, we can now use the expectation value

$$\begin{aligned}
\mathrm{Cov}(\tilde{X}) = \mathrm{Cov}(AX + b) &= \mathrm{E}\left[\left((AX + b) - (A\mu + b)\right)\left((AX + b) - (A\mu + b)\right)^\top\right] \\
&= \mathrm{E}\left[\left(AX - A\mu\right)\left(AX - A\mu\right)^\top\right] \\
&= \mathrm{E}\left[A\left(X - \mu\right)\left(X - \mu\right)^\top A^\top\right] \\
&= A\mathrm{E}\left[\left(X - \mu\right)\left(X - \mu\right)^\top\right] A^\top \\
&= AKA^\top
\end{aligned} \tag{A.5}$$

■

A.1.1 Marginal of a Gaussian Distribution

In this section, we are interested in marginalizing a multivariate Gaussian over a subset of its variables, so for example, for the two dimensional Gaussian that is shown in Figure A.1, we are interested to find the marginal distribution of only one dimension. More formally, we are interested in the distribution, that results if we integrate over a subset of variables of a multivariate Gaussian. Let us define by

$$x = \begin{pmatrix} x_a \\ x_b \end{pmatrix} \tag{A.6}$$

a disjoint partition of a random vector x. For the example above, we could define a and b as the subsets that hold the first and second dimension of x. Additionally, we can also split mean and covariance such that

$$\mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} \qquad K = \begin{pmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{pmatrix} . \tag{A.7}$$

Concretely, we are interested in the marginal

$$p(x_a) = \int p(x_a, x_b) \mathrm{d}x_b \tag{A.8}$$

which results from integrating (summing for a discrete distribution) over all outcomes of x_b. However, instead of solving the integral, we can also use the affine transformation rule that was presented in Lemma A.1.

Lemma A.2. *Let $X \sim \mathcal{N}(\mu, K)$ be a random variable where it's mean and covariance are partitioned as above. Then, the marginal $p(x_a)$ has mean and covariance*

$$p(x_a) = \mathcal{N}(\mu_a, K_{aa}) \tag{A.9}$$

Proof. The resulting marginal seems intuitive, we can now use the affine transformation rule to prove it. Let us set $b = 0$ and define a matrix A such that:

$$A = \begin{pmatrix} B & C \end{pmatrix} \tag{A.10}$$

where $B = \mathrm{diag}(1)$ is a diagonal matrix that has ones on the diagonal, and zero everywhere else, and C is simply a matrix filled with zeros. Then we can compute $AX + b$ and using the affine transformation rule, we know what the mean will look like:

$$\mathrm{E}(AX + b) = A\mu + b = A\mu = \mu_A \tag{A.11}$$

and the covariance

$$\mathrm{Cov}(AX + b) = AKA^\top = K_{aa} \tag{A.12}$$

as $AKA\top = K_{aa}$ is certainly not singular, the proof is complete. ■

A.1.2 Conditional of a Gaussian Distribution

In this section, we are interested in determining the conditional distribution of some variables given some other variables, where we know that their joint probability is Gaussian. More formally, we assume that

$$X \sim \mathcal{N}(\mu, K) \tag{A.13}$$

and we have a again a disjoint partition of x such as

$$x = \begin{pmatrix} x_a \\ x_b \end{pmatrix} \tag{A.14}$$

and then, we want to determine what the distribution of $p(x_a \mid x_b)$ looks like. As we will see, the conditional is also a Gaussian, therefore we only have to find it's mean $\mu_{a\mid b}$ and covariance $K_{a\mid b}$ or precision $P_{a\mid b}$. For our application, x_a will resemble the performance of an unseen hyperparameter configuration, where x_b will be a vector containing all the already observed performances. Then, it of course makes sense to infer the conditional $p(x_a \mid x_b)$ to make a prediction of how well the new hyperparameter configuration will perform, as well as how certain we are about it. To determine the conditional, we first also partition our mean and covariance (and precision) such that

$$\mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} \qquad K = \begin{pmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{pmatrix} \qquad P = \begin{pmatrix} P_{aa} & P_{ab} \\ P_{ba} & P_{bb} \end{pmatrix} . \tag{A.15}$$

As we know that our covariance is a symmetric matrix, using $K^\top = K$, we see that $K_{ab}^\top = K_{ba}$, the same also holds for the precision matrix, $P_{ab}^\top = P_{ba}$, as the inverse of a symmetric matrix is still symmetric.

Lemma A.3. *Let $X \sim \mathcal{N}(\mu, K)$ and let $x = (x_a, x_b)$ be a disjoint partition. Then, the conditional $p(x_a \mid x_b)$ is a Gaussian distribution with mean $\mu_{a\mid b}$ and covariance $K_{a\mid b}$*

$$p(x_a \mid x_b) = \mathcal{N}\left(\mu_{a\mid b}, K_{a\mid b}\right) \tag{A.16}$$

where

$$\begin{aligned} \mu_{a\mid b} &= \mu_a - K_{ab} K_{bb}^{-1} (\mu_b - x_b) \\ K_{a\mid b} &= K_{aa} - K_{ab} K_{bb}^{-1} K_{ba} \end{aligned} \tag{A.17}$$

Proof. Since we are interested in estimating $p(x_a \mid x_b)$, let us first use the chain rule of probability to express it as:

$$p(x_a \mid x_b) = \frac{p(x_a, x_b)}{p(x_b)} \tag{A.18}$$

We will check the term in the exponential function of the conditional, which is obtained by dividing the joint density by the marginal density of x_b which is known from Lemma

A.2 from the previous subsection. The exponential term of the joint density $p(x_a, x_b)$ can be expanded to:

$$
\begin{aligned}
-\frac{1}{2}(x-\mu)^\top K^{-1}(x-\mu) &= -\frac{1}{2}(x-\mu)^\top \begin{pmatrix} P_{aa} & P_{ab} \\ P_{ba} & P_{bb} \end{pmatrix} (x-\mu) \\
&= -\frac{1}{2}(x_a-\mu_a)^\top P_{aa}(x_a-\mu_a) - \frac{1}{2}(x_a-\mu_a)^\top P_{ab}(x_b-\mu_b) \\
&\quad -\frac{1}{2}(x_b-\mu_b)^\top P_{ba}(x_a-\mu_a) - \frac{1}{2}(x_b-\mu_b)^\top P_{bb}(x_b-\mu_b)
\end{aligned} \tag{A.19}
$$

whereas the exponential term in the marginal $p(x_b)$ is simply

$$
-\frac{1}{2}(x_b-\mu_b)^\top P_{bb}(x_b-\mu_b) \ . \tag{A.20}
$$

Dividing both densities, we have to compute the difference of both exponential terms, so effectively we are adding the second term to the first and observe that the last term, introduced ony by $p(x_b)$ is being cancelled out. Therefore, we can continue expanding terms to obtain

$$
\begin{aligned}
-\frac{1}{2}(x-\mu)^\top K^{-1}(x-\mu) &= -\frac{1}{2}(x_a-\mu_a)^\top P_{aa}(x_a-\mu_a) - \frac{1}{2}(x_a-\mu_a)^\top P_{ab}(x_b-\mu_b) \\
&\quad -\frac{1}{2}(x_b-\mu_b)^\top P_{ba}(x_a-\mu_a) \\
&= -\frac{1}{2}x_a^\top P_{aa}x_a + \frac{1}{2}x_a^\top P_{aa}\mu_a + \frac{1}{2}\mu_a^\top P_{aa}x_a - \frac{1}{2}\mu_a^\top P_{aa}\mu_a \\
&\quad -\frac{1}{2}x_a^\top P_{ab}x_b + \frac{1}{2}x_a^\top P_{ab}\mu_b + \frac{1}{2}\mu_a^\top P_{ab}x_b - \frac{1}{2}\mu_a^\top P_{ab}\mu_b \\
&\quad -\frac{1}{2}x_b^\top P_{ba}x_a + \frac{1}{2}x_b^\top P_{ba}\mu_a + \frac{1}{2}\mu_b^\top P_{ba}x_a - \frac{1}{2}\mu_b^\top P_{ba}\mu_a \\
&= -\frac{1}{2}x_a^\top P_{aa}x_a + x_a^\top P_{aa}\mu_a - x_a^\top P_{ab}x_b + x_a^\top P_{ab}\mu_b + C \\
&= -\frac{1}{2}x_a^\top P_{aa}x_a + x_a^\top \left(P_{aa}\mu_a + P_{ab}(\mu_b - x_b)\right) + C \ .
\end{aligned} \tag{A.21}
$$

The first equality simply expands all terms, the second transformation holds by using the fact that $P_{ab}^\top = P_{ba}$ and subsuming all terms that are independent of x_a in one constant C. We can easily see that this term is quadratic in x_a, and now use equation A.2 to find mean and covariance (precision) of the conditional. The first term reveals that the covariance is simply

$$
K_{a|b} = P_{aa}^{-1} \tag{A.22}
$$

and for the mean we get

$$\begin{aligned}\mu_{a|b} &= K_{a|b}(P_{aa}\mu_a + P_{ab}(\mu_b - x_b)) \\ &= P_{aa}^{-1}(P_{aa}\mu_a + P_{ab}(\mu_b - x_b)) \\ &= \mu_a + P_{aa}^{-1}P_{ab}(\mu_b - x_b)\end{aligned} \tag{A.23}$$

However, this mean and covariance depend on parts of the precision matrix, specifically on P_{aa} and P_{ab} which we may not have at hand. Therefore, to express these moments using the covariance matrix, we can use the following identity for the inverse of a block partitioned matrix

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} (A - BD^{-1}C)^{-1} & -(A - BD^{-1}C)^{-1}BD^{-1} \\ -D^{-1}C(A - BD^{-1}C)^{-1} & D^{-1} + D^{-1}C(A - BD^{-1}C)^{-1}BD^{-1} \end{pmatrix} \tag{A.24}$$

which allows us to compute P_{aa} using the definition of the precision

$$\begin{pmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{pmatrix}^{-1} = \begin{pmatrix} P_{aa} & P_{ab} \\ P_{ba} & P_{bb} \end{pmatrix} \tag{A.25}$$

so that

$$\begin{aligned}P_{aa} &= (K_{aa} - K_{ab}K_{bb}^{-1}K_{ba})^{-1} \\ P_{ab} &= -(K_{aa} - K_{ab}K_{bb}^{-1}K_{ba})^{-1}K_{ab}K_{bb}^{-1} \quad .\end{aligned} \tag{A.26}$$

Using the result for P_{aa} and substituting it into Equation A.22 yields

$$K_{a|b} = K_{aa} - K_{ab}K_{bb}^{-1}K_{ba} \tag{A.27}$$

and the mean of the conditional can be computed by substituting P_{aa} and P_{ab} into Equation A.23 which leads to

$$\begin{aligned}\mu_{a|b} &= \mu_a + P_{aa}^{-1}P_{ab}(\mu_b - x_b) \\ &= \mu_a - (K_{aa} - K_{ab}K_{bb}^{-1}K_{ba})(K_{aa} - K_{ab}K_{bb}^{-1}K_{ba})^{-1}K_{ab}K_{bb}^{-1}(\mu_b - x_b) \\ &= \mu_a - K_{ab}K_{bb}^{-1}(\mu_b - x_b)\end{aligned} \tag{A.28}$$

which completes the proof. ■

As already stated above, we will use the conditional Gaussian when making inference with Gaussian processes, where x_a will be the performance of an unknown hyperparameter configuration, conditioned on already observed hyperparameter configuration

performances which are stored in x_b. The results of this section tells us that the conditional is Gaussian, and how to compute it's mean and covariance - or variance as in the one dimensional case we will be interested in. We can already see that both the mean and the variance require inverting the covariance matrix of the observed instances, which is the computation where most time will be used.

A.1.3 Product of Gaussian Distributions

In this subsection, we will look at products of Gaussians, with one slight limitation. As we will only be interested in one-dimensional Gaussians, we will only discuss products of one dimensional Gaussians. The results are employed in Chapter 6, where we build surrogate models based on many individually learned Gaussian processes.

Lemma A.4. *Let $f(x)$ and $g(x)$ be two Gaussian probability density functions with means μ_f and μ_g and variance σ_f^2 and σ_g^2 respectively such that their densities are Gaussian*

$$f(x) = \frac{1}{2\sqrt{\pi}\sigma_f}\exp\left(\frac{-(x-\mu_f)^2}{2\sigma_f^2}\right) \qquad g(x) = \frac{1}{2\sqrt{\pi}\sigma_g}\exp\left(\frac{-(x-\mu_g)^2}{2\sigma_g^2}\right) \tag{A.29}$$

Then, the product follows a Gaussian with mean μ_{fg} and variance σ_{fg}^2

$$f(x)g(x) \sim \mathcal{N}(\mu_{fg}, \sigma_{fg}^2) \tag{A.30}$$

where μ_{fg} and σ_{fg}^2 are given by:

$$\mu_{fg} = \frac{\mu_f\sigma_g^2 + \mu_g\sigma_f^2}{\sigma_f^2 + \sigma_g^2} \qquad \sigma_{fg}^2 = \sqrt{\frac{\sigma_f^2\sigma_g^2}{\sigma_f^2 + \sigma_g^2}} \tag{A.31}$$

Proof. As always, let us multiply the two Gaussians and then inspect their exponential term, to find if it is quadratic in x. By multiplying both terms, the exponential term becomes

$$\begin{aligned} \alpha &= \frac{(x-\mu_f)^2}{2\sigma_f^2} + \frac{(x-\mu_g)^2}{2\sigma_g^2} \\ &= \frac{x^2 - 2x\mu_f + \mu_f^2}{2\sigma_f^2} + \frac{x^2 - 2x\mu_g + \mu_g^2}{2\sigma_g^2} \end{aligned} \tag{A.32}$$

where we simply have expanded both of the quadratic terms. Now, we put both terms on the same denominator

$$\begin{aligned}\alpha &= \frac{\sigma_g^2(x^2 - 2x\mu_f + \mu_f^2) + \sigma_f^2(x^2 - 2x\mu_g + \mu_g^2)}{2\sigma_f^2\sigma_g^2} \\ &= \frac{(\sigma_f^2 + \sigma_g^2)x^2 - 2(\mu_f\sigma_g^2 + \mu_g\sigma_f^2)x + \mu_f^2\sigma_g^2 + \mu_g^2\sigma_f^2}{2\sigma_f^2\sigma_g^2} \\ &= \frac{x^2 - 2\frac{(\mu_f\sigma_g^2 + \mu_g\sigma_f^2)}{\sigma_f^2+\sigma_g^2}x + \frac{\mu_f^2\sigma_g^2 + \mu_g^2\sigma_f^2}{\sigma_f^2+\sigma_g^2}}{2\frac{\sigma_f^2\sigma_g^2}{\sigma_f^2+\sigma_g^2}}\end{aligned} \tag{A.33}$$

which is a quadratic term in x. We can now complete the square by adding a constant that does not depend on x. Given the above equation and comparing it to a general one dimensional Gaussian, we can see that the mean is

$$\mu_{fg} = \frac{\mu_f\sigma_g^2 + \mu_g\sigma_f^2}{\sigma_f^2 + \sigma_g^2} \tag{A.34}$$

and the variance is

$$\sigma_{fg}^2 = \sqrt{\frac{\sigma_f^2\sigma_g^2}{\sigma_f^2 + \sigma_g^2}} \tag{A.35}$$

which completes the proof. ■

In our application, we will likely be interested in not only the product of two Gaussians, but the product of arbitrarily many Gaussians. We therefore have to expand the previous lemma to the product of n Gaussians.

Lemma A.5. *Let $f_i(x)$ be Gaussian densities for all $i = 1,...,n$, such that*

$$f_i(x) = \frac{1}{2\sqrt{\pi}\sigma_i}\exp\left(\frac{-(x-\mu_i)^2}{2\sigma_i^2}\right) \quad \forall i = 1,...,n \tag{A.36}$$

Then, the product of all of these Gaussians

$$f(x) = \prod_{i=1}^{n} f_i(x) \sim \mathcal{N}(\mu_f, \sigma_f^2) \tag{A.37}$$

where mean and variance are given as:

$$\mu_f = \sigma_f^2 \sum_{i=1}^{n} \sigma_i^{-2}\mu_i \qquad \sigma_f^2 = \sum_{i=1}^{n} \sigma_i^{-2} \tag{A.38}$$

Proof. The proof is rather straightforward via induction over n. For $n = 2$, the induction beginning has already been shown in the above lemma. Also for the induction step - going from n to $n+1$ - the transformations necessary have been shown above. We can simply introduce new variables to omit some of the sum notation, and then, in the final step open up these sums to include the additional terms. Out of these reasons, we omit to state the whole proof as this does not show anything new. ■

BIBLIOGRAPHY

[1] C. ANDRIEU, N. DE FREITAS, A. DOUCET, AND M. I. JORDAN, *An introduction to mcmc for machine learning*, Machine learning, 50 (2003), pp. 5–43.

[2] B. BAKER, O. GUPTA, R. RASKAR, AND N. NAIK, *Accelerating neural architecture search using performance prediction*, (2018).

[3] R. BARDENET, M. BRENDEL, B. KEGL, AND M. SEBAG, *Collaborative hyperparameter tuning*, in Proceedings of the 30th International Conference on Machine Learning (ICML-13), S. Dasgupta and D. Mcallester, eds., vol. 28, JMLR Workshop and Conference Proceedings, May 2013, pp. 199–207.

[4] R. BATTITI, *Accelerated backpropagation learning: Two optimization methods*, Complex systems, 3 (1989), pp. 331–342.

[5] M. BELKIN, P. NIYOGI, AND V. SINDHWANI, *Manifold regularization: A geometric framework for learning from labeled and unlabeled examples*, Journal of machine learning research, 7 (2006), pp. 2399–2434.

[6] J. BENNETT, S. LANNING, ET AL., *The netflix prize*, in Proceedings of KDD cup and workshop, vol. 2007, New York, NY, USA, 2007, p. 35.

[7] J. BERGSTRA AND Y. BENGIO, *Random search for hyper-parameter optimization*, J. Mach. Learn. Res., 13 (2012), pp. 281–305.

[8] C. M. BISHOP ET AL., *Pattern recognition and machine learning*, vol. 4, springer New York, 2006.

[9] L. BOTTOU, *Large-scale machine learning with stochastic gradient descent*, in Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.

[10] Y. CAO AND D. J. FLEET, *Generalized product of experts for automatic and principled fusion of gaussian process predictions*, arXiv preprint arXiv:1410.7827, (2014).

[11] O. CHAPELLE, V. VAPNIK, AND Y. BENGIO, *Model selection for small sample regression*, Machine Learning, 48 (2002), pp. 9–23.

[12] T. CHEN AND C. GUESTRIN, *Xgboost: A scalable tree boosting system*, in Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, ACM, 2016, pp. 785–794.

[13] F. CHOLLET ET AL., *Keras*. `https://keras.io`, 2015.

[14] G. CORANI, A. BENAVOLI, J. DEMVAR, F. MANGILI, AND M. ZAFFALON, *Statistical comparison of classifiers through bayesian hierarchical modelling*, Machine Learning, 106 (2017), pp. 1817–1837.

[15] C. CORTES AND V. VAPNIK, *Support-vector networks*, Machine learning, 20 (1995), pp. 273–297.

[16] B. F. DE SOUZA, A. C. DE CARVALHO, R. CALVO, AND R. P. ISHII, *Multiclass svm model selection using particle swarm optimization*, in Hybrid Intelligent Systems, 2006. HIS'06. Sixth International Conference on, IEEE, 2006, pp. 31–31.

[17] M. P. DEISENROTH AND J. W. NG, *Distributed gaussian processes*, in International Conference on Machine Learning (ICML), vol. 2, 2015, p. 5.

[18] T. DOMHAN, J. T. SPRINGENBERG, AND F. HUTTER, *Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves.*, in IJCAI, vol. 15, 2015, pp. 3460–8.

[19] T. DOZAT, *Incorporating nesterov momentum into adam*, (2016).

[20] L. R. DRUMOND, E. DIAZ-AVILES, L. SCHMIDT-THIEME, AND W. NEJDL, *Optimizing multi-relational factorization models for multiple target relations*, in Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM '14, New York, NY, USA, 2014, ACM, pp. 191–200.

[21] J. DUCHI, E. HAZAN, AND Y. SINGER, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research, 12 (2011), pp. 2121–2159.

[22] V. A. EPANECHNIKOV, *Non-parametric estimation of a multivariate probability density*, Theory of Probability & Its Applications, 14 (1969), pp. 153–158.

[23] B. EVERITT AND T. HOTHORN, *An introduction to applied multivariate analysis with R*, Springer Science & Business Media, 2011.

[24] J. R. FINKEL, A. KLEEMAN, AND C. D. MANNING, *Efficient, feature-based, conditional random field parsing*, Proceedings of ACL-08: HLT, (2008), pp. 959–967.

[25] C.-S. FOO, C. B. DO, AND A. Y. NG, *Efficient multiple hyperparameter learning for log-linear models*, in Advances in neural information processing systems, 2008, pp. 377–384.

[26] Y. FREUND AND R. E. SCHAPIRE, *A decision-theoretic generalization of on-line learning and an application to boosting*, Journal of computer and system sciences, 55 (1997), pp. 119–139.

[27] M. FRIEDMAN, *The use of ranks to avoid the assumption of normality implicit in the analysis of variance*, Journal of the american statistical association, 32 (1937), pp. 675–701.

[28] M. FRIEDMAN, *A comparison of alternative tests of significance for the problem of m rankings*, The Annals of Mathematical Statistics, 11 (1940), pp. 86–92.

[29] F. FRIEDRICHS AND C. IGEL, *Evolutionary tuning of multiple svm parameters*, Neurocomput., 64 (2005), pp. 107–117.

[30] R. GARNETT, *Lecture notes on bayesian optimization*.
`https://www.cse.wustl.edu/~garnett/cse515t/spring_2015/files/lecture_notes/12.pdf`.
Accessed: 2018-06-05.

[31] M. HALL, E. FRANK, G. HOLMES, B. PFAHRINGER, P. REUTEMANN, AND I. H. WITTEN, *The weka data mining software: an update*, ACM SIGKDD explorations newsletter, 11 (2009), pp. 10–18.

[32] R. HECHT-NIELSEN, *Theory of the backpropagation neural network*, in Neural networks for perception, Elsevier, 1992, pp. 65–93.

[33] P. HENNIG AND C. J. SCHULER, *Entropy search for information-efficient global optimization*, Journal of Machine Learning Research, 13 (2012), pp. 1809–1837.

[34] G. HINTON, *A practical guide to training restricted boltzmann machines*, Momentum, 9 (2010), p. 926.

[35] G. E. HINTON, *Products of experts*, in Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470), vol. 1, IET, 1999, pp. 1–6.

[36] G. E. HINTON AND D. VAN CAMP, *Keeping the neural networks simple by minimizing the description length of the weights*, in Proceedings of the sixth annual conference on Computational learning theory, ACM, 1993, pp. 5–13.

[37] A. E. HOERL AND R. W. KENNARD, *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics, 12 (1970), pp. 55–67.

[38] C.-J. HSIEH, K.-W. CHANG, C.-J. LIN, S. S. KEERTHI, AND S. SUNDARARAJAN, *A dual coordinate descent method for large-scale linear svm*, in Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 408–415.

[39] F. HUTTER, H. H. HOOS, AND K. LEYTON-BROWN, *Sequential model-based optimization for general algorithm configuration*, in Proceedings of the 5th International Conference on Learning and Intelligent Optimization, LION'05, Berlin, Heidelberg, 2011, Springer-Verlag, pp. 507–523.

[40] T. JOACHIMS, *Svmlight: Support vector machine*, SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund, 19 (1999).

[41] D. R. JONES, *A taxonomy of global optimization methods based on response surfaces*, Journal of global optimization, 21 (2001), pp. 345–383.

[42] D. R. JONES, M. SCHONLAU, AND W. J. WELCH, *Efficient global optimization of expensive black-box functions*, J. of Global Optimization, 13 (1998), pp. 455–492.

[43] R. A. JONSON AND D. W. WICHERN, *Applied multivariate statistical analysis*, 1992.

[44] A. KAPOOR, H. AHN, Y. QI, AND R. W. PICARD, *Hyperparameter and kernel learning for graph based semi-supervised classification*, in Advances in Neural Information Processing Systems, 2005, pp. 627–634.

[45] S. S. KEERTHI, V. SINDHWANI, AND O. CHAPELLE, *An efficient method for gradient-based adaptation of hyperparameters in svm models*, in Advances in neural information processing systems, 2007, pp. 673–680.

[46] M. G. KENDALL, *A New Measure of Rank Correlation*, Biometrika, 30 (1938), pp. 81–93.

[47] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[48] A. KLEIN, S. FALKNER, J. T. SPRINGENBERG, AND F. HUTTER, *Learning curve prediction with bayesian neural networks*, ICLR, (2017).

[49] Y. KOREN, *Factorization meets the neighborhood: a multifaceted collaborative filtering model*, in Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2008, pp. 426–434.

[50] Y. KOREN, R. BELL, AND C. VOLINSKY, *Matrix factorization techniques for recommender systems*, Computer, 42 (2009).

[51] H. W. KUHN AND A. W. TUCKER, *Nonlinear programming*, in Traces and emergence of nonlinear programming, Springer, 2014, pp. 247–258.

[52] H. J. KUSHNER, *A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise*, Journal of Basic Engineering, 86 (1964), pp. 97–106.

[53] H. LAROCHELLE, D. ERHAN, A. COURVILLE, J. BERGSTRA, AND Y. BENGIO, *An empirical evaluation of deep architectures on problems with many factors of variation*, in Proceedings of the 24th international conference on Machine learning, ACM, 2007, pp. 473–480.

[54] D. C. LIU AND J. NOCEDAL, *On the limited memory bfgs method for large scale optimization*, Mathematical programming, 45 (1989), pp. 503–528.

[55] Y. LOU, *Mltk: Machine learning toolkit*. `http://www.cs.cornell.edu/~yinlou/projects/mltk/`.

[56] Z. LUO AND G. WAHBA, *Hybrid adaptive splines*, Journal of the American Statistical Association, 92 (1997), pp. 107–116.

[57] D. J. MACKAY, *Information-based objective functions for active data selection*, Neural computation, 4 (1992), pp. 590–604.

[58] D. J. MacKay, *Probable networks and plausible predictions - a review of practical bayesian methods for supervised neural networks*, Network: Computation in Neural Systems, 6 (1995), pp. 469–505.

[59] T. Masada, D. Fukagawa, A. Takasu, T. Hamada, Y. Shibata, and K. Oguri, *Dynamic hyperparameter optimization for bayesian topical trend analysis*, in Proceedings of the 18th ACM Conference on Information and knowledge management, ACM, 2009, pp. 1831–1834.

[60] T. May, *Influence of binary mask estimation errors on robust speaker identification*, Speech Communication, 87 (2017), pp. 40–48.

[61] A. D. McQuarrie and C.-L. Tsai, *Regression and time series model selection*, World Scientific, 1998.

[62] J. Mockus, *Application of bayesian approach to numerical methods of global and stochastic optimization*, Journal of Global Optimization, 4 (1994), pp. 347–365.

[63] J. Mockus, V. Tiesis, and A. Zilinskas, *Toward global optimization, volume 2, chapter bayesian methods for seeking the extremum*, (1978).

[64] K. P. Murphy, *Machine learning: a probabilistic perspective*, MIT press, 2012.

[65] R. M. Neal, *Bayesian learning for neural networks*, vol. 118, Springer Science & Business Media, 2012.

[66] P. Nemenyi, *Distribution-free multiple comparisons*, in Biometrics, vol. 18, INTERNATIONAL BIOMETRIC SOC 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, 1962, p. 263.

[67] B. T. Polyak, *Some methods of speeding up the convergence of iteration methods*, USSR Computational Mathematics and Mathematical Physics, 4 (1964), pp. 1–17.

[68] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*, The MIT Press, 2005.

[69] S. Rendle, *Factorization machines*, in Data Mining (ICDM), 2010 IEEE 10th International Conference on, IEEE, 2010, pp. 995–1000.

[70] S. RENDLE, C. FREUDENTHALER, Z. GANTNER, AND L. SCHMIDT-THIEME, *Bpr: Bayesian personalized ranking from implicit feedback*, in Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence, AUAI Press, 2009, pp. 452–461.

[71] S. RENDLE, C. FREUDENTHALER, AND L. SCHMIDT-THIEME, *Factorizing personalized markov chains for next-basket recommendation*, in Proceedings of the 19th international conference on World wide web, ACM, 2010, pp. 811–820.

[72] S. RENDLE AND L. SCHMIDT-THIEME, *Pairwise interaction tensor factorization for personalized tag recommendation*, in Proceedings of the third ACM international conference on Web search and data mining, ACM, 2010, pp. 81–90.

[73] H. ROBBINS AND S. MONRO, *A stochastic approximation method*, The annals of mathematical statistics, (1951), pp. 400–407.

[74] L. ROSASCO, E. D. VITO, A. CAPONNETTO, M. PIANA, AND A. VERRI, *Are loss functions all the same?*, Neural Computation, 16 (2004), pp. 1063–1076.

[75] R. SALAKHUTDINOV AND A. MNIH, *Bayesian probabilistic matrix factorization using markov chain monte carlo*, in Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 880–887.

[76] T. SCHAUL, S. ZHANG, AND Y. LECUN, *No more pesky learning rates*, in International Conference on Machine Learning, 2013, pp. 343–351.

[77] N. SCHILLING, T. WINDLER, AND L. SCHMIDT-THIEME, *Hyperparameter optimization across problem tasks*, in Archives of Data Science, 2018.
currently being reviewed.

[78] N. SCHILLING, M. WISTUBA, L. DRUMOND, AND L. SCHMIDT-THIEME, *Hyperparameter optimization with factorized multilayer perceptrons*, in Machine Learning and Knowledge Discovery in Databases, Springer, 2015, pp. 87–103.

[79] N. SCHILLING, M. WISTUBA, L. DRUMOND, AND L. SCHMIDT-THIEME, *Joint model choice and hyperparameter optimization with factorized multilayer perceptrons*, in Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on, IEEE, 2015, pp. 72–79.

[80] N. SCHILLING, M. WISTUBA, AND L. SCHMIDT-THIEME, *Scalable hyperparameter optimization with products of gaussian process experts*, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2016, pp. 33–48.

[81] P. SCHNEIDER, M. BIEHL, AND B. HAMMER, *Hyperparameter learning in probabilistic prototype-based models*, Neurocomputing, 73 (2010), pp. 1117–1124.

[82] M. SEEGER AND O. CHAPELLE, *Cross validation optimization for structured hessian kernel methods*, Submitted. See www. kyb. tuebingen. mpg. de/bs/people/seeger, (2006).

[83] C. E. SHANNON, *A mathematical theory of communication*, ACM SIGMOBILE Mobile Computing and Communications Review, 5 (2001), pp. 3–55.

[84] G. F. SMITS AND E. M. JORDAAN, *Improved svm regression using mixtures of kernels*, in Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on, vol. 3, IEEE, 2002, pp. 2785–2790.

[85] A. J. SMOLA AND P. L. BARTLETT, *Sparse greedy gaussian process regression*, in Advances in neural information processing systems, 2001, pp. 619–625.

[86] J. SNOEK, H. LAROCHELLE, AND R. P. ADAMS, *Practical bayesian optimization of machine learning algorithms*, in Advances in Neural Information Processing Systems 25, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds., Curran Associates, Inc., 2012, pp. 2951–2959.

[87] N. SRIVASTAVA, G. HINTON, A. KRIZHEVSKY, I. SUTSKEVER, AND R. SALAKHUTDINOV, *Dropout: A simple way to prevent neural networks from overfitting*, The Journal of Machine Learning Research, 15 (2014), pp. 1929–1958.

[88] K. SWERSKY, J. SNOEK, AND R. P. ADAMS, *Freeze-thaw bayesian optimization*, arXiv preprint arXiv:1406.3896, (2014).

[89] R. TIBSHIRANI, *Regression shrinkage and selection via the lasso: a retrospective*, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 73 (2011), pp. 273–282.

[90] A. N. TIKHONOV, A. GONCHARSKY, V. STEPANOV, AND A. G. YAGOLA, *Numerical methods for the solution of ill-posed problems*, vol. 328, Springer Science & Business Media, 2013.

[91] V. TRESP, *A bayesian committee machine*, Neural Computation, 12 (2000), pp. 2719–2741.

[92] J. VANSCHOREN, J. N. VAN RIJN, B. BISCHL, AND L. TORGO, *Openml: networked science in machine learning*, ACM SIGKDD Explorations Newsletter, 15 (2014), pp. 49–60.

[93] D. S. WATKINS, *Fundamentals of matrix computations*, vol. 64, John Wiley & Sons, 2004.

[94] M. WISTUBA, N. SCHILLING, AND L. SCHMIDT-THIEME, *Two-stage transfer surrogate model for automatic hyperparameter optimization*, in Joint European conference on machine learning and knowledge discovery in databases, Springer, 2016, pp. 199–214.

[95] M. WISTUBA, N. SCHILLING, AND L. SCHMIDT-THIEME, *Scalable gaussian process-based transfer surrogates for hyperparameter optimization*, Machine Learning, (2018), pp. 1–36.

[96] D. YOGATAMA AND G. MANN, *Efficient transfer learning method for automatic hyperparameter tuning*, in International Conference on Artificial Intelligence and Statistics (AISTATS 2014), 2014.

[97] M. D. ZEILER, *Adadelta: an adaptive learning rate method*, arXiv preprint arXiv:1212.5701, (2012).